Tipo char.

Primero describiremos los **valores que pueden tomar** los elementos de tipo char.

Es un tipo básico del lenguaje. Las variables y constantes de tipo char ocupan un byte.
El tipo unsigned char tiene el rango 0 a 255.
El tipo char (o signed char, esto es por defecto) tiene el rango –128 a 127.
A las variables de tipo char se las puede tratar como si fueran de tipo entero, ya que son convertidas automáticamente a ese tipo cuando aparecen en expresiones.

Una constante de tipo carácter se define como un carácter encerrado entre comillas simples. El valor de una constante de tipo carácter es el valor numérico de ese carácter en la tabla o código de caracteres. En la actualidad la tabla más empleada es el código ASCII.

Definición de variables y constantes de tipo char.

char ch; declara una variable de tipo char.
ch = ‘c’; asigna la constante de tipo carácter c a la variable ch.

ASCII son las iniciales de American Standard Code for Information Interchange.
Internamente, un carácter, se representa por una secuencia binaria de 8 bits. Un valor perteneciente al código ASCII es la representación numérica de un carácter como '1' o '@' o de una acción de control.

Se tienen 32 caracteres de control, que no son imprimibles o visualizables. En general puede especificarse un carácter por su valor numérico equivalente expresado en octal, mediante '\ooo' donde una, dos o las tres o deben ser reemplazadas por un dígito octal (dígitos entre 0 y 7). La secuencia binaria de 8 unos seguidos, equivale a 377 en octal.

Alternativamente pueden emplearse dos cifras hexadecimales para representar un carácter, del siguiente modo: '\xhh' La x indica que uno o los dos dígitos siguientes deben ser reemplazados por una cifra hexadecimal (dígitos 0 a 9, y las letras A, B, C, D, F). La secuencia binaria de 8 unos seguidos, equivale a FF en hexadecimal.

H		D	H		D	H		D	H		D	H		D	H		D	H		D	H		D
00	NULL	00	10	DEL	16	20		32	30	0	48	40	@	64	50	P	80	60	`	96	70	p	112
01	SOH	01	11	DC1	17	21	!	33	31	1	49	41	A	65	51	Q	81	61	a	97	71	q	113
02	STX	02	12	DC2	18	22	"	34	32	2	50	42	B	66	52	R	82	62	b	98	72	r	114
03	EXT	03	13	DC3	19	23	#	35	33	3	51	43	C	67	53	S	83	63	c	99	73	s	115
04	EOT	04	14	DC4	20	24	$	36	34	4	52	44	D	68	54	T	84	64	d	100	74	t	116
05	ENQ	05	15	NAK	21	25	%	37	35	5	53	45	E	69	55	U	85	65	e	101	75	u	117
06	ACK	06	16	SYN	22	26	&	38	36	6	54	46	F	70	56	V	86	66	f	102	76	v	118
07	BEL	07	17	ETB	23	27	'	39	37	7	55	47	G	71	57	W	87	67	g	103	77	w	119
08	BS	08	18	CAN	24	28	(	40	38	8	56	48	H	72	58	X	88	68	h	104	78	x	120
09	TAB	09	19	EM	25	29	)	41	39	9	57	49	I	73	59	Y	89	69	i	105	79	y	121
0a	LF	10	1a	SUB	26	2a	*	42	3a	:	58	4a	J	74	5a	Z	90	6a	j	106	7a	z	122
0b	VT	11	1b	ESC	27	2b	+	43	3b	;	59	4b	K	75	5b	[	91	6b	k	107	7b	{	123
0c	FF	12	1c	FS	28	2c	,	44	3c	<	60	4c	L	76	5c	\	92	6c	l	108	7c	|	124
0d	CR	13	1d	GS	29	2d	-	45	3d	=	61	4d	M	77	5d	]	93	6d	m	109	7d	}	125
0e	SO	14	1e	RS	30	2e	.	46	3e	>	62	4e	N	78	5e	^	94	6e	n	110	7e	~	126
0f	SI	15	1f	US	31	2f	/	47	3f	?	63	4f	O	79	5f	_	95	6f	o	111	7f	del	127

Todos los valores de la tabla anterior son positivos, si se representan con un byte.

Los caracteres que representan los dígitos decimales tienen valores asociados menores que las letras; y si se les resta 0x30, los cuatro bits menos significativos representan a los dígitos decimales en BCD (Binary Coded Decimal). Las letras mayúsculas tienen códigos crecientes en orden alfabético, y son menores en 0x20 que las letras minúsculas.

En español suelen emplearse los siguientes caracteres, que se anteceden por su equivalente decimal: 130 é, 144 É, 154 Ü, 160 á, 161 í, 162 ó, 163 ú, 164 ñ, 165 Ñ, 168 ¿, 173 ¡.
Los valores de éstos tienen el octavo bit (el más significativo en uno), y forman parte de los 128 caracteres que conforman un código ASCII extendido.

Los caracteres de control han sido designados por tres letras que son las primeras del significado de la acción que tradicionalmente e históricamente se les ha asociado.

Por ejemplo el carácter FF (Form Feed) con valor 0x0c, se lo emplea para enviar a impresoras, y que éstas lo interpreten con la acción de avanzar el papel hasta el inicio de una nueva página (esto en impresoras que son alimentadas por formularios continuos).

Los teclados pueden generar caracteres de control (oprimiendo la tecla control y una letra). Por ejemplo ctrl-S y ctrl-Q generan DC3 y DC1 (también son conocidos por Xon y Xoff), y han sido usados para detener y reanudar largas salidas de texto por la pantalla de los terminales). Varios de los caracteres se han usado en protocolos de comunicación, otros para controlar modems.

Algunos de los caracteres, debido a su uso, tienen una representación por secuencias de escape. Se escriben como dos caracteres, pero representan el valor de uno de control. Los más usados son:

\n representa a nueva línea (new line o line feed).
En Unix esto genera un carácter de control, en archivos de texto en PC, se generan dos: 0x0D seguido de 0x0A.
\t tabulador horizontal.
\0 Nul representa el carácter con valor cero. El que se emplea como terminador de string.

El siguiente texto,
se representa internamente
según:

45 6C 20 73 69 67 75 69 65 6E 74 65 20 74 65 78 74 6F 2C 20 0D 0A
73 65 20 72 65 70 72 65 73 65 6E 74 61 20 69 6E 74 65 72 6E 61 6D 65 6E 74 65 20 0D 0A
73 65 67 FA 6E 3A 0D 0A

La representación hexadecimal de los caracteres que forman el texto, muestra los dos caracteres de control que representan el fin de línea (0x0D seguido de 0x0A). Cada carácter gráfico es representado por su valor numérico hexadecimal. La primera representación (externa) se emplea para desplegar la información en pantallas e impresoras; la segunda es una representación interna (se suele decir binaria, pero representada en hexadecimal) y se emplea para almacenar en memoria o en medios magnéticos u ópticos.

Si se representan como constantes de tipo char, se definen entre comillas simples.
Por ejemplo:
#define EOS '\0' /* End of string */

Estas secuencias de escape pueden incorporarse dentro de strings. El \n (backslash n) suele aparecer en el string de control de printf, para denotar que cuando se lo encuentre, debe cambiarse de línea en el medio de salida.

Dentro de un string, suelen emplearse las siguientes secuencias para representar los caracteres ", ', \. Que no podrían ser usados ya que delimitan strings o caracteres o son parte de la secuencia de escape.
\\ para representar la diagonal invertida
\" para representar la comilla doble, dentro del string.
\' para representar la comilla simple dentro del string.

Ejemplo:
Char esc = '\\';
"O\'Higgins" en un string.

Expresiones.

Un carácter en una expresión es automáticamente convertido a entero.

Así entonces la construcción de expresiones que involucren variables o constantes de tipo carácter son similares a las que pueden plantearse para enteros. Sin embargo las construcciones más frecuentes son las comparaciones.

La especificación de tipo char es signed char.

Puede verificarse cómo son tratados los enteros con signo negativo por el compilador que se está empleando, observando los resultados de: printf(" %c\n",-23); Debe producir la letra acentuada: é.

Un compilador moderno también debería imprimir las letras acentuadas, por ejemplo:

```
printf(" %c\n",'é');
```

El siguiente par de for anidados muestra 8 renglones de 16 caracteres cada uno, con los caracteres que tienen valores negativos(el bit más significativo del byte es uno).

```
for (i = -128; i<0; i++)
  { for (j = 0; j<16; j++) printf("%c ", i++); putchar('\n'); }
```

Observando la salida, la que dependerá del compilador empleado, pueden comprobarse los caracteres (con valores negativos) que serán representados gráficamente.

Cuando se desea tratar el contenido de un byte (independiente del valor gráfico) debe emplearse unsigned char.

Si se desea obtener el entero i que es representado por un carácter c, conviene emplear:
i = c –'0'; en lugar de: i = c - 48; o i = c - 0x30; ya que el código generado resulta independiente del set de caracteres (ASCII, EBCDIC). La primera expresión asume que los valores de los caracteres, que representan dígitos decimales, están asociados a enteros consecutivos y ordenados en forma ascendente.

Es preferible escribir: i = (int) (c – '0'); que destaca que se está efectuando un conversión de tipos. Pero no suele encontrarse en textos escritos por programadores experimentados.

Si se desea comparar si el carácter c es igual a determinada constante, conviene la expresión: (c == 'b') en lugar de (c == 98).

La expresión ('a' - 'A') toma valor (97 – 65) = 32. Valor que expresado en binario es:
00100000 y en hexadecimal 0x20.

Si con esta máscara se efectúa un or con una variable c de tipo carácter: c | ('a' - 'A')
la expresión resultante queda con el bit en la quinta posición en uno(esto conviniendo, como es usual, que el bit menos significativo ocupa la posición cero, el más derechista). La expresión: c |= ('a' - 'A') si c es una letra la convierte en letra minúscula.

La expresión: c & ~ ('a' - 'A') forma un and con la máscara binaria 11011111 y la expresión resultante deja un cero en el bit en la quinta posición.

La palabra máscara recuerda algo que se pone delante del rostro, y en este caso es una buena imagen de la operación que realiza. Nótese que con un or con una máscara pueden setearse determinadas posiciones con unos; y con un and, con una máscara, pueden dejarse determinados bits en cero.

Las expresiones anteriores pueden emplearse para convertir letras minúsculas a mayúsculas y viceversa.

for(c='a'; c<='z'; c++) { bloque }
permite ejecutar repetidamente un bloque de acciones, variando c desde la a hasta la z cada vez que se realiza el bloque.
Esto asume que las letras recorridas en orden alfabético, están ordenadas en una secuencia numérica ascendente.
En la parte de incremento del for, la conversión automática a entero no requiere un casteo explícito: ((int) c)++

La condición:
(c != ' ' && c != '\t' && c!= '\n')
es verdadera(toma valor 1) si c no es un separador.
Por De Morgan, la condición:
(c == ' ' || c != '\t' || c!= '\n') es verdadera si c es separador.

Más adelante se ilustran otros ejemplos de expresiones de tipo char.

Entrada-Salida

En el lenguaje Pascal la entrada y salida son acciones. En C, son invocaciones a funciones o macros de la biblioteca estándar.
Las siguientes descripciones son abstracciones (simplificaciones) del código que efectivamente se emplea.

int **putchar** (int c)
{ return putc (c, stdout); }

Convierte c a unsigned char y lo escribe en la salida estándar (stdout); retorna el carácter escrito, o EOF, en caso de error.

int **getchar** (void)
{ return getc (stdin); }

Lee desde la entrada estándar el siguiente carácter como unsigned char y lo retorna convertido a entero; si encuentra el fin de la entrada o un error retorna EOF.

La entrada estándar normalmente es el teclado, y la salida la pantalla. Sin embargo pueden redirigirse la entrada y la salida desde o hacia archivos.

El valor de la constante EOF predefinida en <stdio.h> es un valor entero, distinto a los valores de los caracteres que pueden desplegarse en la salida, suele ser –1. EOF recuerda a end of file (debió ser fin del stream o flujo de caracteres). Por esta razón getchar, retorna entero, ya que también debe detectar el EOF.

La condición: ((c = getchar( )) != EOF) con c de tipo int.

obtiene un carácter y lo asigna a c (la asignación es una expresión que toma el valor del lado izquierdo); este valor es comparado con el EOF, si son diferentes, la condición toma valor 1, que se interpreta como valor verdadero. Los paréntesis son obligatorios debido a que la precedencia de != es mayor que la del operador =.

Sin éstos, la interpretación sería:
c = (getchar( ) != EOF), lo cual asigna a c sólo valores 0 ó 1.

Para copiar el flujo de entrada hacia la salida, puede escribirse:

```
while ((c = getchar( )) != EOF) putchar(c);
```

Para contar en n las veces que se presenta el carácter t, puede escribirse:

```
while ((c = getchar( )) != EOF) if (c == t ) n++;
```

Entrada y salida con formato.

La siguiente función ilustra el uso de printf para caracteres:

```
void print_char (unsigned char c)
{
  if (isgraph(c)) printf("'%c'\n", c); else printf("'\\%.3o'\n",  c);
}
```

Dentro del string de control del printf una especificación de conversión se inicia con el carácter % y termina con un carácter. El carácter c indica que una variable de tipo int o char se imprimirá como un carácter. Nótese que los caracteres que están antes y después de la especificación de conversión se imprimen de acuerdo a su significado.

Los siguientes valores del argumento actual son imprimibles (isgraph retorna verdadero).
print_char(0x40); imprime en una línea: '@'
print_char(65); imprime en una línea: 'A'

Los siguientes valores del argumento actual se imprimen como tres cifras octales.
print_char(06); imprime en una línea: '\006'
print_char(0x15); imprime en una línea: '\025'

Entre el % y el carácter de conversión a octal o, pueden encontrarse otros especificadores:
- especifica ajuste a la izquierda.
n.m donde n es el número del ancho mínimo del campo y m el número máximo de caracteres que será impreso. Si el campo es más ancho, se rellena con espacios.

Si el string de control del printf que se ejecuta asociado al else se modifica a:
`"'\\x%.2x'\n"` Se pasa hacia la salida '\x y luego dos caracteres hexadecimales debido al carácter de conversión x.
Con esta modificación:
print_char('\n'); imprime en una línea: '\x0a'
print_char(0x15); imprime en una línea: '\x15'

Debe notarse que el ancho y el máximo número de caracteres se refieren a la variable o expresión que será convertida.
printf("'%4.1c'\n", 'A' +1) imprime '   B'
printf("'%-4.1c'\n",'3' -1) imprime '2   '

Funciones.
Las siguientes funciones transforman letras minúsculas a mayúsculas y viceversa.

```
char toupper(register int c)
{
  if((char)c <= 'z' && (char)c >= 'a')     c &= ~('a' - 'A');
  return (char)c;
}

char tolower(int c)
{
  if((char)c <= 'Z' && (char)c >= 'A')      c |= ('a' - 'A');
  return (char)c;
}
```

La siguiente es una función que retorna verdadero si el carácter es imprimible:

```
int isgraph (int c)
{
  return((unsigned char)c >= ' ' && (unsigned char)c <= '~');
}
```

La siguiente es una función que retorna verdadero si el carácter es un dígito decimal:

```
int isdig(int c)
{
  return((unsigned char)c >= '0' && (unsigned char)c <= '9');
}
```

La función recursiva **printd** imprime un número decimal.

Primero el signo, luego la cifra más significativa. Lo logra reinvocando a la función con un argumento que trunca, mediante división entera la última cifra. De este modo la primera función (de las múltiples encarnaciones) que termina, es la que tiene como argumento n a la cifra más significativa, que es menor que 10, imprimiendo dicho valor, a través de la conversión de entero a carácter. Es necesario tomar módulo 10, para imprimir las cifras siguientes.

Igual resultado se logra con: printf("%d", n).

```
 void printd(int n)
{  if (n<0) putchar('-') n= -n;
  if(n/10) printd(n/10);
  putchar(n % 10 + '0');
}
```

La función prtbin imprime en binario un entero de 16 bits.

```
void prtbin(int i)
{  int j;
   for (j=15; j>=0; j--) if(i &(1<<j) ) putchar('1'); else putchar('0');
}
```

La función prtstr imprime un string. Igual resultado se logra con printf("%s", s).

```
void prtstr(char *s)
{
  while(*s) putchar(*s++);
}
```

La traducción a assembler de putchar ocupa alrededor de 10 instrucciones, y la utilización de printf emplea algunos cientos de instrucciones. Algunas de las rutinas anteriores pueden ser útiles cuando no se dispone de una gran cantidad de memoria, como en el caso de microcontroladores.

Macros.

Es un mecanismo que permite el reemplazo de símbolos en el texto fuente. Lo efectúa el preprocesador antes de iniciar la compilación.

Pueden ser con y sin argumentos formales.
Cuando no se emplean argumentos, permite asignar un valor a una constante. Esto puede emplearse para mejorar la legibilidad de un programa.

Se escriben:
#define <token> <string>

Todas las ocurrencias del identificador <token> en el texto fuente serán reemplazadas por el texto definido por <string>. Nótese que no hay signo igual, y que no se termina con punto y coma.

También las emplea el sistema para representar convenientemente y en forma estándar algunos valores.
Por ejemplo en limits.h figuran entre otras definiciones, las siguientes:

```
#define CHAR_BIT        8
#define SCHAR_MAX       127
#define SCHAR_MIN       (-128)
#define UCHAR_MAX       255
```

En float.h, se definen entre otras:

```
#define DBL_MIN         2.2250738585072014E-308
#define FLT_MIN         1.17549435E-38F
```

En ctype.h, se encuentran entre otras:

```
#define _IS_SP      1       /* is space */
#define _IS_DIG     2       /* is digit indicator */
#define _IS_UPP     4       /* is upper case */
#define _IS_LOW     8       /* is lower case */
#define _IS_HEX    16       /* [0..9] or [A-F] or [a-f] */
#define _IS_CTL    32       /* Control */
#define _IS_PUN    64       /* punctuation */
```

En values.h, se encuentran entre otras:

```
#define MAXINT          0x7FFF
#define MAXLONG         0x7FFFFFFFL
#define MAXDOUBLE       1.797693E+308
#define MAXFLOAT        3.37E+38
#define MINDOUBLE       2.225074E-308
```

Si se desean emplear constantes predefinidas por el sistema, debe conocerse en cual de los archivos del directorio include están definidas. Y antes de que sean usadas debe indicarse en una línea la orden de inclusión, para que el preprocesador incorpore el texto completo de ese archivo en el texto fuente previo al proceso de compilación.
Por ejemplo:

```
#include  <ctype.h>
```

Los paréntesis de ángulo indican que el archivo está ubicado en el subdirectorio include. Si se desea tener archivos definidos por el usuario, el nombre del archivo que debe incluirse debe estar encerrado por comillas dobles.

Macros con argumentos.
Se suelen emplear para definir macroinstrucciones (de eso deriva su nombre), es decir una expresión en base a las acciones primitivas previamente definidas por el lenguaje. La macro se diferencia de una función en que no incurre en el costo de invocar a una función (crear un frame con espacio para los argumentos y variables locales en el stack, salvar registros y la dirección de retorno; y luego recuperar el valor de los registros salvados, desarmar el frame y seguir la ejecución).

Su real efectividad está limitada a situaciones en las que el código assembler que genera es pequeño en comparación con el costo de la administración de la función equivalente. O también cuando se requiere mayor velocidad de ejecución, no importando el tamaño del programa ejecutable.

```
#define <macro> ( <arg1>, ... )  <string>
```

Los identificadores arg1, ... son tratados como parámetros del macro. Todas las instancias de los argumentos son reemplazadas por el texto definido para arg1,.. cuando se invoca a la macro, mediante su nombre. Los argumentos se separan por comas.

Por ejemplo:
```
#define ISLOWER(c) ('a' <= (c) )&& (c) <= 'z')
#define TOUPPER(c) (ISLOWER(c) ? 'A' + ((c) - 'a') : (c))
```

Si en el texto fuente aparece ISLOWER('A') dentro de una expresión, antes de la compilación el preprocesador cambia el texto anterior por: ('a' <= ('A') && ('A') <= 'z'). El objetivo de esta macro es devolver un valor verdadero (valor numérico 1) si el carácter c tiene un valor numérico entre los valores del código asociados al carácter 'a' y al carácter 'z'; en caso contrario, la expresión lógica toma valor falso (valor numérico 0).

Si aparece TOUPPER('d') éste es reemplazado por el texto:
```
(('a' <= ('d') && ('d') <= 'z') ? 'A' + (('d') - 'a') : ('d'))
```
TOUPPER convierte a mayúsculas un carácter ASCII correspondiente a una letra minúscula.

La definición debe estar contenida en una línea. En caso de que el string sea más largo, se emplea el carácter \ al final de la línea.
```
#define ctrl(c) ((c) \
< ' ')          /* string continua desde línea anterior */
```

Lo cual es equivalente a:

```
#define isctrl(c) ((c) < ' ')
```
Macro que toma valor 1 si el carácter c es de control.

El siguiente macro sólo puede aplicarse si se está seguro que el argumento representa un carácter que es una letra. Entre 0x41 y 0x51. También da resultado correcto si la letra es minúscula (entre 0x60 y 0x7a).

```
#define TOLOWER(c)    ( (c) | 0x20)
```

Otros ejemplos de macros:
```
#define  isascii(c)   ( !( (c) &~0x7F))
#define  toascii(c)   ( (c) & 0x7F)
```

Nótese que en el string que define el texto que reemplaza al macro, los argumentos se colocan entre paréntesis.

Biblioteca. ctype.c

Prototipos en include/ctype.h

El diseño de la biblioteca ctype se efectúa mediante una tabla de búsqueda, en la cual se emplean macros para clasificar un carácter. La tabla es un arreglo de bytes(unsigned char) en los que se codifica en cada bit una propiedad del carácter asociado.

El nombre de cada macro pregunta si el carácter es de cierta clase, por ejemplo si es símbolo alfanumérico el nombre es isalnum. Cada macro retorna un valor diferente de cero en caso de ser verdadero y cero en caso de ser falso.

Se define el concepto asociado a cada bit.

```
#define  _U     0x01   /* Upper. Mayúsculas */
#define  _L     0x02   /* Lower. Minúsculas */
#define  _N     0x04   /* Número decimal    */
#define  _S     0x08   /*  Espacio          */
#define  _P     0x10   /*  Puntuación       */
#define  _C     0x20   /* Control           */
#define  _X     0x40   /*  Hex              */
```

En funciones de biblioteca, se suelen preceder los indentificadores por un _(underscore o línea de subrayado); de este modo se evita el alcance de nombres con identificadores definidos por el usuario (siempre que éste no emplee como primer símbolo para sus identificadores el subrayado _).

El siguiente arreglo contiene la información de atributos de cada carácter, indexada por su valor númerico asccii +1.

```
const unsigned char _ctype_[129] = {
    0,                                      /*retorna falso para EOF  */
  _C, _C, _C, _C, _C, _C, _C,  _C,
  _C, _C| _S, _C| _S, _C| _S,  _C| _S,  _C| _S,  _C, _C,   /* bs, tab, lf, vt, ff, cr, so, si  */
  _C,  _C,  _C,  _C,  _C,  _C,  _C,  _C,
  _C,  _C,  _C,  _C,  _C,  _C,  _C,  _C,
  _S,  _P,  _P,  _P,  _P,  _P,  _P,  _P,          /* space, !, ", #, $, %, &, '    */
  _P,  _P,  _P,  _P,  _P,  _P,  _P,  _P,                  /* (, ), *, +,  ,  , -, . , /              */
  _N,  _N,  _N,  _N,  _N,  _N,  _N,  _N,          /* 0, 1, 2, 3, 4, 5, 6, 7   */
  _N,  _N,  _P,  _P,  _P,  _P,  _P,  _P,          /* 8, 9, :, ;, <, =, >, ?    */
  _P,  _U|_X,  _U|_X,  _U|_X,  _U|_X,  _U|_X,  _U|_X,  _U,  /*@,A,B,C,D,E,F,G */
  _U,  _U,  _U,  _U,  _U,  _U,  _U,  _U,          /* H, I, J, K, L, M, N, O    */
  _U,  _U,  _U,  _U,  _U,  _U,  _U,  _U,          /* P, Q, R, S, T, U, V, W  */
  _U,  _U,  _U,  _P,  _P,  _P,  _P,  _P,          /* X, Y, Z, [, \, ], [ , ^, _     */
  _P,  _L|_X,  _L|_X,  _L|_X,  _L|_X,  _L|_X, _L|_X, _L,   /* `, a, b, c, d, e, f, g  */
```

```
  _L,  _L,  _L,  _L,  _L,  _L,  _L,  _L,  /* h, i, j, k, l, m, n, o   */
  _L,  _L,  _L,  _L,  _L,  _L,  _L,  _L,  /* p, q, r, s, t, u, v, w  */
  _L,  _L,  _L,  _P,  _P,  _P,  _P,  _C   /*  x, y, z, {, |, }, ~,  DEL   */
};
```

Si el valor entero del carácter es –1(EOF), al sumarle 1, resulta índice 0 para la tabla de búsqueda. Si se buscan los atributos en la entrada 0 del arreglo; se advierte que tiene definido valor cero, resultando con retornos falsos de los macros para EOF.
Se escoge para EOF un valor numérico diferente a los imprimibles.

isascii está definida para valores enteros. El resto de los macros están definidos sólo cuando isaccii es verdadero o si c es EOF.

```
#define     isascii(c)     ( (unsigned)(c) < 128)
#define     iscntrl(c)     (_ctype_[(unsigned char) (c) + 1]&_C)
#define     isupper(c)     (_ctype_[(unsigned char) (c) + 1]&_U)
#define     islower(c)     (_ctype_[(unsigned char) (c) + 1]&_L)
#define     isalpha(c)     (_ctype_[(unsigned char) (c) + 1]&(_U | _L))
#define     isdigit(c)     (_ctype_[(unsigned char) (c) + 1]&_N)
#define     isxdigit(c)    (_ctype_[(unsigned char) (c) + 1]&(_N | _X))
#define     isalnum(c)     (_ctype_[(unsigned char) (c) + 1]&(_U | _L | _N))
#define     isspace(c)     (_ctype_[(unsigned char) (c) + 1]&_S)
#define     ispunct(c)     (_ctype_[(unsigned char) (c) + 1]&_P)
#define     isprint(c)     (_ctype_[(unsigned char) (c) + 1]&(_P | _U | _L | _N | _S))
#define     isgraph(c)     (_ctype_[(unsigned char) (c) + 1]&(_P | _U | _L | _N))
```

Los caracteres considerados gráficos no contemplan la categoría espacio (_S); pero sí están considerados en la de imprimibles.
En la categoría espacios se consideran los caracteres de control: tab, lf, vt, ff, cr
Las letras de las cifras hexadecimales se consideran en mayúsculas y minúsculas.

Con el macro siguiente, que pone en uno el bit en posición dada por bit:

```
# define        _ISbit(bit)     (1 << (bit))
```

La definición de atributos puede efectuarse según:

```
#define  _U     _ISbit(0)      /* Upper. Mayúsculas */
#define  _L     _ISbit(1)      /* Lower. Minúsculas */
#define  _N     _ISbit(2)      /* Número decimal    */
#define  _S     _ISbit(3)      /*  Espacio           */
#define  _P     _ISbit(4)      /*  Puntuación        */
#define  _C     _ISbit(5)      /* Control           */
#define  _X     _ISbit(6)      /*  Hex               */
```

Los códigos de biblioteca suelen ser más complejos que los ilustrados, ya que consideran la portabilidad. Por ejemplo si el macro que define un bit en determinada posición de una palabra, se desea usar en diferente tipo de procesadores, se agrega texto alternativo de acuerdo a la característica.

Las órdenes de compilación condicional seleccionan, de acuerdo a las condiciones, la parte del texto fuente que será compilada.

Por ejemplo: Si se desea marcar uno de los bits de una palabra de 16 bits, el macro debe considerar el orden de los bytes dentro de la palabra.

```
# if __BYTE_ORDER == __BIG_ENDIAN
# define _ISbit(bit)       (1 << (bit))
# else /* __BYTE_ORDER == __LITTLE_ENDIAN */
# define _ISbit(bit)       ((bit) < 8 ? ((1 << (bit)) << 8) : ((1 << (bit)) >> 8))
# endif
```