# Índice :

| | |
|---|---|
| **isalnum**<br>**isalpha**<br>**isascii**<br>**iscntrl**<br>**isdigit**<br>**isgraph**<br>**islower**<br>**isprint**<br>**ispunct**<br>**isspace**<br>**isupper**<br>**isxdigit** | <ctype.h> |

Prototipos :

```
. int isalnum(int car);
. int isalpha(int car);
. int isascii(int car);
. int iscntrl(int car);
. int isdigit(int car);
. int isgraph(int car);
. int islower(int car);
. int isprint(int car);
. int ispunct(int car);
. int isspace(int car);
. int isupper(int car);
. int isxdigit(int car);
```

Observaciones :

. Estas son macros de clasificación de carácteres, que clasifican los valores enteros de los números de ASCII mediante acceso a tablas . Cada macro se resuelve devolviendo un valor distinto de cero si es verdad o cero si es falso .

. **isascii** está definida para todos los valores enteros .

. Las restantes macros de la familia, sólo cuando **car** cumple con que **isascii** o **car** es **EOF** (-1), para cualquier otro valor el resultado es impredecible .

. Se puede invocar la función correspondiente haciendo #undef de la macro correspondiente .

Valor devuelto :

. Estas macros devuelven un valor distinto de cero en el caso que **car** :

. **isalnum** : esté en el rango **'0'..'9'** ó **'A'..'Z'** ó **'a'..'z'** (alfanumérico) .

. **isalpha** : esté en el rango **'A'..'Z'** ó **'a'..'z'** (alfabético) .

. **isascii** : esté en el rango **0..127** (**0x00..0x7f**) (los 1ros 128 carácteres ASCII) .

. **iscntrl** : sea un cáracter de control o un carácter **del** (**0x00..0x1f ó 0x7f**) .

. **isdigit** : sea un dígito (**'0'..'9'**) .

. **isgraph** : tenga una representación 'gráfica', con excepción del ' ' (**0x21..0x7e**) .

. **islower** : sea el número de ASCII de una minúscula (**'a'..'z'**) .

. **isprint** : sea un carácter imprimible, incluyendo el ' ' (**0x20..0x7e**) .

. **ispunct** : sea un carácter de puntuación : **!"#$%&'()*+,-./:;<=>?@** (**0x21..0x40**), **[\]^_`** (**0x5b..0x60**), **{|}~** (**0x7b..0x7e**) .

. **isspace** : sea un **'\t'**, **'\r'**, **'\v'**, **'\f'**, **'\n'** ó **' '** (**0x09..0x0d, 0x20**) .

. **isupper** : sea una mayúscula (**'A'..'Z'**) .

. **isxdigit**: sea un dígito hexadecimal (**'0'..'9'**, **'A'..'Z'**, **'a'..'z'**) .

| Compatibilidad | DOS | UNIX | ANSI C |
|---|---|---|---|
| **isascii** | Si | Si | |
| **is..** | Si | Si | Si |

Ejemplos :

En primer lugar un programa que crea un archivo de texto con el que se genera un array preinicializado para que las macros/funciones determinen qué devuelven :

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

/* básicos */
#define __IS_cntrl    0x01   /* '\b'(0x7f), 0x00 .. 0x1f                    */
#define __IS_digit    0x02   /* '0' .. '9'                                  */
#define __IS_lower    0x04   /* 'a' .. 'z'                                  */
#define __IS_print    0x08   /* ' '(0x20) .. 0x0e                           */
#define __IS_punct    0x10   /* !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~           */
#define __IS_space    0x20   /* '\t', '\n', '\v', '\f', '\r' y ' '          */
#define __IS_upper    0x40   /* 'A' .. 'Z'                                  */
#define __IS_xdigit   0x80   /* '0' .. '9', 'A' .. 'F', 'a' .. 'f'          */
/* derivados */
#define __IS_alnum    0x46   /* (__IS_digit | __IS_upper | __IS_lower)      */
#define __IS_alpha    0x44   /* (__IS_upper | __IS_lower)                   */
#define __IS_graph    0x56   /* (__IS_punct | __IS_alnum)                   */

void main(void)
{
   int ciclo;
   unsigned char valor;
   FILE *fp;

   if((fp = fopen("visascii.h", "wt")) == NULL)
      exit(printf("Error creando archivo\n"));

   fprintf(fp, "char __c_type[] = { 0x00");
   for(ciclo = 0; ciclo < 128; ciclo++)
   {
      valor = 0;
      if(ciclo < 32 || ciclo == 127)
         valor |= __IS_cntrl;
      if(ciclo >= '0' && ciclo <= '9')
         valor |= __IS_digit;
      if(ciclo >= 'a' && ciclo <= 'z')
         valor |= __IS_lower;
      if(ciclo > ' ' && ciclo < '0'     || ciclo > '9' && ciclo < 'A' ||
         ciclo > 'Z' && ciclo < 'a'     || ciclo > 'z' && ciclo < 127)
         valor |= __IS_punct;
      if(ciclo >= ' ' && ciclo < 0x7f)
         valor |= __IS_print;
      if(ciclo >= '\t' && ciclo <= '\r' || ciclo == ' ')
         valor |= __IS_space;
      if(ciclo >= 'A' && ciclo <= 'Z')
         valor |= __IS_upper;
      if(ciclo >= 'A' && ciclo <= 'F'   || ciclo >= 'a' && ciclo <= 'f' ||
         ciclo >= '0' && ciclo <= '9')
         valor |= __IS_xdigit;

      fprintf(fp,
              ",%s0x%02x",
              ciclo && !(ciclo % 8) ? "\n                          " : " ",
              valor);
   }
   fprintf(fp, " };\n");
   fclose(fp);
}
```

El archivo generado contiene :

```
char __c_type[] = { 0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                          0x01, 0x21, 0x21, 0x21, 0x21, 0x21, 0x01, 0x01,
                          0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                          0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                          0x28, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x8a, 0x8a, 0x8a, 0x8a, 0x8a, 0x8a, 0x8a, 0x8a,
                          0x8a, 0x8a, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x18, 0xc8, 0xc8, 0xc8, 0xc8, 0xc8, 0xc8, 0x48,
                          0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48,
                          0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48,
                          0x48, 0x48, 0x48, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x18, 0x8c, 0x8c, 0x8c, 0x8c, 0x8c, 0x8c, 0x0c,
                          0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
                          0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
                          0x0c, 0x0c, 0x0c, 0x18, 0x18, 0x18, 0x18, 0x01 };
```

El primer valor almacenado en el array (posición 0) corresponde a **car = -1 (EOF)**, el segundo (posición 1) a **car = 0**, y así sucesivasmente hasta **car = 128**, con lo que hay 129 valores almacenados .

Se completa el contenido de este archivo con las definiciones y funciones correspondientes, creando un programa que verifique si lo obtenido por nosotros coincide con lo que se obtiene con las macros originales :

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

/* array preinicializado para el rango  -1..127 (cuando vale isascii) */
char __c_type[] = { 0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                          0x01, 0x21, 0x21, 0x21, 0x21, 0x21, 0x01, 0x01,
                          0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                          0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
                          0x28, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x8a, 0x8a, 0x8a, 0x8a, 0x8a, 0x8a, 0x8a, 0x8a,
                          0x8a, 0x8a, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x18, 0xc8, 0xc8, 0xc8, 0xc8, 0xc8, 0xc8, 0x48,
                          0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48,
                          0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48,
                          0x48, 0x48, 0x48, 0x18, 0x18, 0x18, 0x18, 0x18,
                          0x18, 0x8c, 0x8c, 0x8c, 0x8c, 0x8c, 0x8c, 0x0c,
                          0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
                          0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
                          0x0c, 0x0c, 0x0c, 0x18, 0x18, 0x18, 0x18, 0x01 };

/* básicos */
#define __IS_cntrl    0x01   /* '\b'(0x7f), 0x00 .. 0x1f                  */
#define __IS_digit    0x02   /* '0' .. '9'                                */
#define __IS_lower    0x04   /* 'a' .. 'z'                                */
#define __IS_print    0x08   /* ' '(0x20) .. 0x0e                         */
#define __IS_punct    0x10   /* !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~         */
#define __IS_space    0x20   /* '\t', '\n', '\v', '\f', '\r' y ' '        */
#define __IS_upper    0x40   /* 'A' .. 'Z'                                */
#define __IS_xdigit   0x80   /* '0' .. '9', 'A' .. 'F', 'a' .. 'f'        */
/* derivados */
#define __IS_alnum    0x46   /* (__IS_digit | __IS_upper | __IS_lower)    */
#define __IS_alpha    0x44   /* (__IS_upper | __IS_lower)                 */
#define __IS_graph    0x56   /* (__IS_punct | __IS_alnum)                 */

/* versiones como funciones */
int is_alnum(int __x)
{
   return __c_type[__x + 1] & __IS_alnum;
}
```

```
int is_alpha(int __x)
{
   return __c_type[__x + 1] & __IS_alpha;
}

int is_ascii(int __x)
{
   return (unsigned)__x < 128;
}

int is_cntrl(int __x)
{
   return __c_type[__x + 1] & __IS_cntrl;
}

int is_digit(int __x)
{
   return __c_type[__x + 1] & __IS_digit;
}

int is_graph(int __x)
{
   return __c_type[__x + 1] & __IS_graph;
}

int is_lower(int __x)
{
   return __c_type[__x + 1] & __IS_lower;
}

int is_print(int __x)
{
   return __c_type[__x + 1] & __IS_print;
}

int is_punct(int __x)
{
   return __c_type[__x + 1] & __IS_punct;
}

int is_space(int __x)
{
   return __c_type[__x + 1] & __IS_space;
}

int is_upper(int __x)
{
   return __c_type[__x + 1] & __IS_upper;
}

int is_xdigit(int __x)
{
   return __c_type[__x + 1] & __IS_xdigit;
}

/* declaración de macro reemplazos */
#define is_alnum(X)  (__c_type[(X) + 1] & __IS_alnum)
#define is_alpha(X)  (__c_type[(X) + 1] & __IS_alpha)

#define is_ascii(X)  ((unsigned)(X) < 128)

#define is_cntrl(X)  (__c_type[(X) + 1] & __IS_cntrl)
#define is_digit(X)  (__c_type[(X) + 1] & __IS_digit)
#define is_graph(X)  (__c_type[(X) + 1] & __IS_graph)
#define is_lower(X)  (__c_type[(X) + 1] & __IS_lower)
#define is_print(X)  (__c_type[(X) + 1] & __IS_print)
```

```
#define is_punct(X)  (__c_type[(X) + 1] & __IS_punct)
#define is_space(X)  (__c_type[(X) + 1] & __IS_space)
#define is_upper(X)  (__c_type[(X) + 1] & __IS_upper)
#define is_xdigit(X) (__c_type[(X) + 1] & __IS_xdigit)

/* en el caso que quiera emplear la función debe hacer #undef, p. ej. :  */
#undef is_cntrl

void main(void)
{
   int ciclo;
   /* isascii definido para cualquier entero vale para  -1 .. 127 */
   for(ciclo = -32767; ciclo < 32767; ciclo++)
      if(isascii(ciclo) != is_ascii(ciclo))
         printf("isascii :%4d\t", ciclo);
   /* comprobacion de los restantes macroreemplazos */
   for(ciclo = -1; ciclo < 128; ciclo++)
   {
/*
 * como seguramente hemos inicializado el array __c_type con valores
 * distintos de los empleados por su compilador, si estas funciones o
 * macros devuelven distinto de 0 (cero) se toma 1 (uno), si devuelven
 * 0 (cero) se toma 0 (cero), tanto para la funcion/macro de biblioteca
 * como para la que hemos construido nosotros
 * con esto, si una devuelve verdad y la otra devuelve falso se informa
 * con un mensaje de error con qué numero de ASCII no coinciden
 */
      if((isspace(ciclo) ? 1 : 0) != (is_space(ciclo) ? 1 : 0))
         printf("isspace :%4d\t", ciclo);
      if((isdigit(ciclo) ? 1 : 0) != (is_digit(ciclo) ? 1 : 0))
         printf("isdigit :%4d\t", ciclo);
      if((isupper(ciclo) ? 1 : 0) != (is_upper(ciclo) ? 1 : 0))
         printf("isupper :%4d\t", ciclo);
      if((islower(ciclo) ? 1 : 0) != (is_lower(ciclo) ? 1 : 0))
         printf("islower :%4d\t", ciclo);
      if((isalnum(ciclo) ? 1 : 0) != (is_alnum(ciclo) ? 1 : 0))
         printf("isalnum :%4d\t", ciclo);
      if((isalpha(ciclo) ? 1 : 0) != (is_alpha(ciclo) ? 1 : 0))
         printf("isalpha :%4d\t", ciclo);
      if((iscntrl(ciclo) ? 1 : 0) != (is_cntrl(ciclo) ? 1 : 0))
         printf("iscntrl :%4d\t", ciclo);
      if((isgraph(ciclo) ? 1 : 0) != (is_graph(ciclo) ? 1 : 0))
         printf("isgraph :%4d\t", ciclo);
      if((isprint(ciclo) ? 1 : 0) != (is_print(ciclo) ? 1 : 0))
         printf("isprint :%4d\t", ciclo);
      if((ispunct(ciclo) ? 1 : 0) != (is_punct(ciclo) ? 1 : 0))
         printf("ispunct :%4d\t", ciclo);
      if((isxdigit(ciclo) ? 1 : 0) != (is_xdigit(ciclo) ? 1 : 0))
         printf("isxdigit :%4d\t", ciclo);
   }
}
```

| **toascii**<br>**tolower**<br>**toupper** | <ctype.h> |
|---|---|

Prototipos :
- **int toascii(int car);**
- **int tolower(int ch);**
- **int toupper(int ch);**

Observaciones :
. **toascii** (macro) convierte el entero **car** al rango 0..127, poniendo a cero todos los bits excepto los 7 de menor peso .
. **tolower** (función) convierte su argumento **car** (en el rango –1..255) a su correspondiente minúscula si corresponde, de lo contrario, devuelve lo que recibió .
. **toupper** (función) convierte su argumento **car** (en el rango –1..255) a su correspondiente mayúscula si corresponde, de lo contrario, devuelve lo que recibió .

Valor Devuelto :
. **toascii** devuelve el valor de **car** convertido a ASCII .
. **tolower** devuelve el número de ASCII de la minúscula si recibe una mayúscula, de lo contrario devuelve lo que recibe .
. **toupper** devuelve el número de ASCII de la mayúscula si recibe una minúscula, de lo contrario devuelve lo que recibe .

| Compatibilidad | DOS | UNIX | ANSI C |
|---|---|---|---|
| **toascii** | Si | Si | |
| **tolower** | Si | Si | Si |
| **toupper** | Si | Si | Si |

Ejemplos :

```
#define to_ascii(X)   ((X) & 0x7f)
```

```
/*
 * a propósito de tolower/toupper :
 * algunos compiladores preveen que :
 *
 * si recibe (-1)
 *    devuelve (-1)
 * si-no
 *    convierte el argumento recibido al rango 0..255
 *    si está en el rango 'A'.. 'Z'(para tolower)/( 'a'.. 'z'(para toupper)
 *       devuelve la minúscula(para tolower)/mayúscula(para toupper)
 *    si-no
 *       devuelve el argumento convertido
 *    fin-si
 * fin-si
 *
 * otros compiladores omiten la conversión al rango 0..255
 */
```

```
int to_lower(int x)
{
   return x == -1 ? x : (x &= 0xff) >= 'A' && x <= 'Z' ? x + 'a' - 'A' : x;
}
```

```
int to_upper(int x)
{
   return x == -1 ? x : (x &= 0xff) >= 'a' && x <= 'z' ? x + 'A' - 'a' : x;
}
```