```
for (i=0;i<10;i++)
     for (j=0;j<10;)
          *(*(x+i)+j++)=(y[i][j]>*(*(z+i)+j))?1:2;

for (;*a|*b;*p++=(*a>*b)?*a++;*b++);
```

Figura 1 Programas crípticos

Identificadores validos	Identificadores no validos	
x	4num	(el primer carácter
y2		(no es letra)
suma_1	"x"	(carácter ilegal ")
_t	orden-no	(carácter ilegal -)
TABLA	ind lis	(espacio ilegal)

Figura 2 Ejemplos de identificadores legales e ilegales

```
!  *  +  \  "  <
#  (  =  |  {  >
%  )  ~  ;  }  /
^  _  [  :  ,  ?
&  -  ]  '  .  (blanco)
```

Figura 3 Caracteres especiales.

```
auto        extern      sizeof
break       float       static
case        for         struct
char        goto        switch
const       if          typedef
continue    int         union
default     long        unsigned
do          register    void
double      return      volatile
else        short       while
enum        signed
```

Figura 4 Palabras reservadas de C

```
char            short int           int
long int        unsigned char       unsigned short int
unsigned int    unsigned long int   float
long floatvoid
```

Figura 5 Tipos de datos fundamentales

char Representa un carácter en código ASCII, también se puede interpretar como un entero.

short int Indica un entero de tamaño corto.

int Entero igual que *integer* en Pascal.

long int Entero largo.

unsigned short int Como *short int* pero sin signo.

unsigned int Como *int* pero sin signo.

unsigned long int Como *long int* pero sin signo.

float Flotante corto. Igual que *single* de Pascal.

double Flotante largo. Idem a *double* de Pascal.

void No indica ningún tipo. Es el tipo de las funciones que no devuelven nada.

Los tipos *short int*, *long int*, *unsigned int* y *long float* se pueden escribir como: *short*, *long*, *unsigned* y *double*.

```
sizeof(char) = 1
sizeof(short) <= sizeof(int) <= sizeof(long)
sizeof(unsigned) = sizeof(int)
sizeof(float) =< sizeof(double)
```

static Cuando se invoca a una función por segunda vez se pierden los valores que las variables locales de la función tenían al acabar la anterior llamada. Declarando una variable de este tipo cuando se llama por segunda vez a la subrutina la variable static (estática) contiene el mismo valor que al acabar la llamada anterior.

auto Es lo mismo que si no se usara ningún modificador

volatile El compilador debe asumir que la variable está relacionada con un dispositivo y que puede cambiar de valor en cualquier momento.

register El compilador procurará almacenar la variable cualificada de este modo en un registro de la CPU.

extern La variable se considera declarada en otro fichero. No se le asignará dirección ni espacio de memoria.

```
struct fecha {
  int mes;
  int día;
  int año;
};                              /* Estructura fecha */
struct cuenta {
  int cuen_no;
  char cuen_tipo;
  float saldo;
  struct fecha ultimo_pago;
} cliente[100];          /* Estructura cuenta y declaración */

struct cuenta clientes[20];
clientes[0].cuen_no = cliente[99].ultimo_pago.año;
```

Figura 6 Ejemplo de struct

```
union palabra {           Se declaran 4 variables
  unsigned char l[2];     a,b,c y d
  unsigned int x;         A cada variable se puede
} a,b,c,d;                acceder como entero (x) o
                          como 2 bytes (l[2]).

a.x = 65535U;        a.x es un entero
a.l[0] = 0;          a.l[0] es un byte
a.l[1] = 255;        a.l[1] es un byte
                     ¿Cuánto vale a.x en este momento?
```

Figura 7 Ejemplo de unión

```
typedef struct {
  int mes;
  int día;
  int año;
} fecha;       /* Declaración de un nuevo tipo llamado fecha */
fecha hoy;     /* Declaración de una variable de tipo fecha */
typedef unsigned char byte;         /* Tipo byte de Pascal */
typedef unsigned int word;          /* Tipo word de Pascal */
```

Figura 8 Ejemplos de typedef

```
enum colores { negro = -1, azul, cian, magenta, rojo = 2, blanco} fondo,
borde;
enum colores primer_plano;
fondo = cian;
```

Figura 9 Ejemplo de enum

```
int a,b,c;                      Tres variables enteras.
float raiz1,raiz2;              Dos variables reales.
char caracter, texto[80];       Un carácter y una string de 80.
short int a;                    Entero corto.
long int b;                     Entero largo.
unsigned short int d;           Entero corto sin signo.
unsigned char a;                Carácter sin signo.
signed char b;                  Carácter con signo.
char texto[3] = "abc";          Declaración e inicialización
char a = '\n';                  Inicialización con Return.
char texto[] = "abc";           Sin especificar tamaño.
extern unsigned short int       Variable externa.
```

Figura 10 Ejemplos de declaración de variables.

sonido (campana)	'\a'
backspace	'\b'
tab horizontal	'\t'
tab vertical	'\v'
nueva línea	'\n'
form feed	'\f'
retorno de carro	'\r'
comillas(")	'\"'
comilla simple(')	'\''
signo interrogación	'\?'
backslash (\)	'\\'
nulo	'\0'

Figura 11 Secuencias de escape.

```
0               Entero
9999            Entero
0x              Entero (hexa)
0x1             Entero (hexa)
0X7FFF          Entero (hexa)
0xabcd          Entero (hexa)
05270           Entero (octal)
50000U          Entero (sin signo)
123456789L      Entero (largo)
123456789UL     Entero (largo y sin signo)
01234L          Entero (octal y largo)
077777UL        Entero (octal, largo y sin signo)
0x456L          Entero (hexa y largo)
0XFFFFUL        Entero (hexa, sin signo y largo)
0.              Flotante (double)
0.2             Flotante (double)
.8              Flotante (double)
2E-2            Flotante (double)
2e3             Flotante (double)
2.2e+5          Flotante (double)
0.12L           Flotante representado con larga precisión
.12e6F          Flotante representado con simple precisión
'A'             Carácter A
'$'             Carácter dolar
'\n'            Carácter carry return
'\x20'          Carácter espacio
"verde"         String
"verde\n"       String que termina con un carry return
"\"RETURN\"\n"  String que contiene RETURN entre comillas     terminado
en
                carry return.
```

Figura 12 Ejemplos de constantes

```
void func(int *pa,int b)
{
  *pa = 1;
  b = 2;
  return ;
}

main()
{
  int a, b;
  a = b = 0;
  func(&a, b);
  /* En este punto ¿cuánto valen a y b? */
  return;
}
```

Figura 13 Ejemplo de paso de punteros a una función

```
int x[100];              Declaración de un array de 100 enteros
x[0]      *x             Primer elemento del array
x[2]      *(x + 2)       Tercer elemento del array
x         x              Dirección del array
&x[3]     (x + 3)        Dirección del tercer elemento del array

char a[] ="Pulsa Return";

a[0]      *a             Carácter "P"
a[i]      *(a + i)       Carácter i-ésimo
&a[0]a                   Dirección de la cadena
```

Figura 14 Ejemplos de punteros y arrays unidimensionales

```
int *px, *py;
px < py
px <= py
px > py
px >= py
px == py
px != py
px == NULL
a = *(px++)          Postincremento del puntero
a = *(++px)          Preincremento del puntero
px - py              "Distancia" entre los punteros px y py
```

Figura 15 Operaciones con punteros

Operaciones aritméticas

```
+          Suma
-          Resta
*          Multiplicación
/          División
++         Incremento
--         Decremento
%          Módulo
```

Operaciones lógicas

```
||         OR lógico
&&         AND lógico
!          NOT lógico
^          XOR lógico
```

Operaciones de bits

```
|          OR
&          AND
~          NOT
>>         Desplazamiento a la derecha
<<         Desplazamiento a la izquierda
```

Comparaciones

```
<          Mayor que
>          Menor que
<=         Menor o igual que
>=         Mayor o igual que
==         Igual que
!=         Distinto que
```

Asignaciones

```
=          Asignación
+=         a=a+3 --> a+=3
-=         a=a-3 --> a-=3
*=         a=a*3 --> a*=3
/=         a=a/3 --> a/=3
%=         a=a%3 --> a%=3
^=         a=a^3 --> a^=3
&=         a=a&3 --> a&=3
|=         a=a|3 --> a|=3
>>=        a=a>>3 --> a>>=3
<<=        a=a<<3 --> a<<=3
```

Operadores especiales

? :	Operador If de expresiones
,	Separador de expresiones
*	Valor apuntado por un puntero
&	Dirección de una variable
(*type*)	Typecasting, donde *type* indica el tipo de datos al que se convierte el resultado de la expresión.
sizeof *type*	Devuelve el número de bytes necesarios para representar un determinado tipo de datos.
.	Cualificador. Separador de campos
->	*(p*).a ==> p->a*

Precedencia de los operadores

```
() [] -> .
! ~ ++ -- - (type) * & sizeof       (todos son unarios)
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?:
= += -= *= /= %= ^= &= |=
,
```

```
int a, b, c;
a = b = c = 0;
a += b = c;
```

Figura 16 Ejemplos de asignaciones

Considérense las declaraciones:

```
char c;        float f;       int i;
short s;       unsigned u;
```

Expresión	Tipo
c – s / i	int
u * 3 – i	unsigned
u * 3.0 – i	double
f * 3 – i	float
c + 1	int
c + 1.0	double

Figura 17 Conversiones implícitas

```
/* Cálculo de la media de un vector */
int v[100], i = 0, media, suma = 0;
while (i < 100) {
  suma += v[i++];
}
media = suma / 100;
```

Figura 18 Ejemplo de instrucción while

```
/* Cálculo de la media de un vector */
int v[100], i, media, suma = 0;
for (i = 0; i < 100; i++) {
  suma += v[i];
}
media = suma / 100;
```

Figura 19 Ejemplo de bucle for

```
                if (estado == 'S')
                  tasa = 0.20 * pago;
                else
                  tasa = 0.14 * pago;

              /* Nótese que en C se escribe ";" antes del else */
```

Figura 20 Ejemplo de sentencia if-else

```
/* Cálculo de la media de un vector */
int v[100], i, media, suma = 0;
i = 0;
do {
  suma += v[i++];
} while (i < 100);
media = suma / 100;
```

Figura 21 do-while

```
char color;
switch (color) {
  case 'a':
  case 'A': printf("AMARILLO\n");
            break;
  case 'r':
  case 'R': printf("ROJO\n");
  case 'b':
  case 'B': printf("BLANCO\n");
  default:printf("OTRO\n");
}
```

Figura 22 switch

```
void funcion1(int a);     /* Prototipo de la función */
void funcion2(void); /* Prototipo de la función */

/* trozo de programa */

void funcion1(int a) /* Función 1 */
{
  /* Cuerpo de la función */
  funcion2();
}

void funcion2(void)  /* Función 2 */
{
  /* Cuerpo de la función */
  funcion1(2);
}
```

Figura 23 Funciones

`#include <stdio.h>`	Incluye la cabecera de las librerias de entrada/salida desde el directorios estándar
`#include "stdio.h"`	Igual al anterior pero las busca en el directorio actual
`#include "a:\librería\mia.h"`	Incluye el fichero mia.h desde la unidad a:

Figura 24 Ejemplo de inclusión de librerías

%c	Carácter
%d	Entero decimal
%e	Flotante se representa con exponente
%f	Flotante se representa sin exponente
%g	Menor entre %e y %f
%o	Entero octal, sin el cero inicial
%u	Entero decimal sin signo
%x	Entero representado en hexa sin 0x
%s	String

Figura 25 Caracteres de conversión

"r"	Abrir un archivo existente solo para lectura
"w"	Abrir un archivo solo para escritura
"a"	Abrir un archivo para añadir. Si no existe se crea uno
"r+"	Abrir un archivo existente para lectura y escritura
"w+"	Abrir un archivo nuevo para lectura y escritura
"a+"	Abrir un archivo nuevo para leer y añadir

Figura 26 Modos de apertura de un fichero

```
#include <stdio.h>

main()
{
FILE *fp;
char a;
int b, c;

if ((fp = fopen("fichero.txt", "r+")) == NULL) {
     printf("ERROR");
     exit(0);
     }
fscanf(fp, "%c %d, %d", &a, &b, &c);
fclose(fp);
return;
}
```

Figura 27 Lectura de datos de un fichero

```
if (a > b)      /* Cálculo del máximo de dos variables */
     c = a;
else
     c = b;

c = (a > b) ? a : b; /* Equivalente con el operador ?: */
```

Figura 28 Operador ?:

```
int funcion1(int a)
{
/* Cuerpo de la función */
}

void funcion2()
{
int a, (*pun_fun)();
pun_fun = funcion1;
a = (*pun_fun)(2);
return ;
}
```

Figura 29 Punteros a funciones

```
/* Típica declaración de la función main() */

main(int argc, char *argv[])
{
/* Cuerpo de la función main */
}
```

Figura 30 Función main()

```
#define NULL 0       /* Definición de constante */

#ifdef DEBUG
  /* Código para depuración */
#else
  /* Código definitivo (sin depuración) */
#endif

#undef EGA /* Quita de la tabla la definición de EGA */
```

Figura 31 Instrucciones del Preprocesador

```
# Fichero Makefile para construir el programa WORDS.EXE
#
# AUTOR:          Kiko
# FECHA:          22.09.94
# FINALIDAD:   Ejemplo de utilización de la utilidad MAKE.
# COMENTARIOS: Para ver las opciones posibles en el compilador de línea,
#                 y en el linker, ejecutar BCC y TLINK respectivamente.
#                 (Ambos programas están en el directorio
#                    \PROG\COMP\BC20\BIN
#                 del servidor de red.
# BIBLIOGRAFIA:  Borland C++ Version 2.0
#    User's Guide
#    Los capítulos correspondientes a la utilidad MAKE y al TLINK
#    están a disposición de los alumnos.
#
# Explicación de las opciones del compilador:
#  -w: Producir todos los Warnings posibles
#  -A: Modo ANSI C
#  -ms: Modelo Small
#  -L: Directorio de las librerías
#  -I: Directorio de los ficheros incluídos
#
# Explicación de las opciones del linker:
#  /v: include full symbolic debug information
#  /n: no default libraries
#  /c: lower case significant in symbols
#  /x: no map file at all

C_OPCIONES=-w -A -ms -Ig:\prog\comp\bc20\include
LINK_OPCIONES=/v /n /c /x
LIB=g:\prog\comp\bc20\lib\

CC=g:\prog\comp\bc20\bin\BCC
LINKER=g:\prog\comp\bc20\bin\TLINK

words.exe: words.obj
 $(LINKER) $(LINK_OPCIONES) $(LIB)\c0s.OBJ words.OBJ, words,,$(LIB)\cs.lib
 del $&.obj

words.obj: words.c
 $(CC) $(C_OPCIONES) -c words.c
```

```
/*********************************************************************
PROGRAMA: wc
AUTOR:      Kiko
FECHA:      12.07.94
FINALIDAD:  Utilidad wc de UNIX: Cuenta líneas, palabras y caracteres
          de un fichero de texto.
COMENTARIOS:Considera palabras cualquier secuencia de caracteres que no
            contenga espacios en blanco, tabuladores o fines de linea
(\n).
HISTORIA:
BIBLIOGRAFIA:  K&R
     "El lenguaje de programación C"
     Kernighan & Ritchie
     Ed. Prentice-Hall Hispanoamericana, S. A.
     pg 20
*********************************************************************/
#include <stdio.h>
#include <stdlib.h>

#define CR   '\r'
#define SPACE  ' '
#define TAB '\t'
#define LF     '\n'

typedef enum {FALSE = 0, TRUE = 1} boolean;

void main(int argc, char *argv[])
{
FILE *f;          /* File handler */
int c,            /* Caracter leído del fichero */
 nl, nw, nc; /* Contadores de líneas, palabras y caracteres */
boolean inword;  /* Indica si se está procesando una palabra */

if (argc < 2) {
 printf("WC  Word  Counter.\nCounts  Lines,  Words  and  Chars  in  a  Text
File\n\n");
 printf("Usage: wc filename\n");
 exit(0);
 }

if ((f = fopen(argv[1], "r")) == NULL) {
 printf("Error trying to open %s as input\n", argv[1]);
 exit(0);
 }

inword = FALSE;
nl = nw = nc = 0;
while ((c = fgetc(f)) != EOF) {
 nc++;
 if (c == CR)
   nl++;
 if (c == SPACE || c == CR || c == TAB || c == LF)
   inword = FALSE;
 else
   if (inword == FALSE) {
     inword = TRUE;
     nw++;
     }
 }
printf("%d %d %d\n", nl, nw, nc);
}
```

```
/* PROGRAMA:p_c_06.c Kiko 13.10.94
FINALIDAD:  Solución al problema 06 de la hoja de problemas de C */

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

#define MAX_LONG 100
char s1[MAX_LONG], s2[MAX_LONG];

/*************************************************************************
/
int slen(char *str) {
   char *s;

   s = str;
   while (*s != '\0')
      s++;
   return(s - str);
}
/*************************************************************************
/
void scat(char *str1, char *str2) {
   char *s = str1, *r = str2;/* Para recorrer las cadenas */

   while (*s != '\0')
      s++;

   while (*r != '\0') {
      *s = *r;
      r++;
      s++;
   }

   *s = '\0';     /* Finalizar la cadena correctamente */
}
/*************************************************************************
/
char *sleft(char *str, int n) {
char *s = str, *mem, *ini;
int longitud;

mem = (char *)malloc((n + 1) * sizeof(char));
if (mem == NULL) {
   printf("Error: No hay memoria suficiente.\n");
   exit(-1);
}
ini = mem;

for (longitud = 0; ((*s != '\0') && (longitud < n)); s++) {
   *mem = *s;
   mem++;
   longitud++;
}
*mem = '\0';
return(ini);
}
```

```
/*************************************************************************
/
char *sright(char *str, int n) {
 char *s;
 char *mem, *ini_cad;
 int num_char;

 ini_cad = str;

 mem = (char *)malloc((n + 1) * sizeof(char));
 if (mem == NULL) {
   printf("Error: No hay memoria suficiente.\n");
   exit(-1);
 }
 mem += (n + 1);

 s = str;
 while (*s != '\0')
   s++;

 ini_cad = str;
 num_char = 0;
 while ((s != ini_cad) && (num_char < n)) {
   *mem = *s;
   mem--;
   s--;
   num_char++;
 }
 *mem = *s;
 return(mem);
}
/*************************************************************************/

void main(void) {

strcpy(s1, "Adiós ");
strcpy(s2, "Mundo cruel");
scat(s1, s2);
printf("s1: %s\n", s1);
printf("3 primeros de s1: %s\n", sleft(s1, 3));
printf("3 últimos de s1: %s\n", sright(s1, 3));
}
```

```
/*********************************************************************
PROGRAMA: files.c Kiko 13.10.94
FINALIDAD:  Ilustrar el uso de ficheros en bajo nivel.
*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <io.h>  /* Librería de ficheros a bajo nivel */
#include <fcntl.h>
#include <sys\stat.h>

#define BUFF_SIZE 10

void main(void) {
int f;    /* File handler */
char *buffer;  /* Puntero al buffer de lectura de bytes */
int bytes;  /* Número de bytes leídos del fichero */
int i;

buffer = (char *)malloc(BUFF_SIZE * sizeof(char));
if (buffer == NULL) {
  printf("Error: No hay memoria.\n");
  exit(-1);
}

f = open("PRUEBA.TXT", O_RDONLY | O_BINARY, S_IWRITE | S_IREAD);

if (f == -1) {
     printf("Error: Imposible abrir el fichero\n");
     exit(1);
 }

while(1) {
  bytes = read(f, buffer, BUFF_SIZE);
  if (bytes == -1) {
    printf("Error leyendo el fichero.\n");
    exit(1);
  }

  for (i = 0; i < bytes; i++)
    printf("%c", buffer[i]);

  if (bytes != BUFF_SIZE)
    break;
}
close(f);
}
```