CAPÍTULO 1

EL PRIMER PROGRAMA EN C. TIPOS DE DATOS

1.1 Salida de datos por pantalla.

1.1.1 El primer programa en C. La función main

Todo programa C consta de una o más funciones. Obligatoriamente, debe contener siempre la función main.

Ejemplo:

```
void main ( )
{

 }
```

Lo que debe escribirse entre las llaves son las instrucciones del programa. En el ejemplo anterior no hay ninguna instrucción, luego el programa no realiza nada, pero es un programa C válido.
La función main no tiene por qué ser la primera de un programa, puede haber instrucciones anteriores. Las definiciones de las funciones adicionales empleadas en el programa pueden preceder o seguir a main.

Un programa C debe escribirse en minúsculas, salvo los identificadores que admiten mayúsculas, tal como se estudia en el apartado correspondiente de este capítulo.

Dependiendo de las instrucciones que vayan a ejecutarse en el programa, puede ser necesario acceder a ciertas librerías de funciones (archivos cabecera) del compilador; éstas deben indicarse al principio del programa después de la expresión #include y delimitadas por ángulos (< >). Cuando se vayan a ejecutar instrucciones de entrada y salida de datos, se debe incluir el archivo cabecera stdio.h, comenzando el programa con la siguiente instrucción:

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

Cuando se ejecuten otras instrucciones (se utilicen otras funciones) se debe incluir la llamada a las librerías correspondientes. Con esto se consigue que el programa al compilar sólo utilice las librerías que necesita, con lo que se reduce el tiempo de compilación y el tamaño del fichero ejecutable resultante.

Una función que se incluye dentro de la biblioteca stdio.h es **printf,** que tiene como objetivo escribir en pantalla.

Veamos a continuación un ejemplo de un programa C que escribe en pantalla la frase: Este es mi primer programa C

Ejemplo:

```
# include <stdio.h>
void main ( )
{
   printf ("Este es mi primer programa C");
}
```

Del análisis del ejemplo anterior se puede extraer que:

- Cada instrucción en C debe terminar con (;).
- Las cadenas de caracteres (texto) en lenguaje C deben escribirse entre comillas dobles.

Para escribir texto en varias líneas se utiliza el símbolo **\n** en cualquier parte del texto a imprimir. Este símbolo equivale a un salto de línea.

Ejemplo:

```
# include <stdio.h>
 void main (  )
 {
    printf ("Esta es una línea de texto \n");
    printf ("Estamos en la ");
    printf ("segunda línea \n \n");
    printf("La anterior línea está en blanco");
 }
```

Es importante resaltar que dos instrucciones printf no equivalen a dos líneas de texto, ya que para saltar de línea está obligado incluir el símbolo \n.

Para imprimir únicamente cadenas de caracteres en pantalla, se puede utilizar la función **puts** (ñcadenaò) que escribe en pantalla la cadena escrita como argumento. Esta función también está incluida en el archivo de cabecera stdio.h. La diferencia entre la función puts y la función printf radica en que puts inserta un salto de línea al finalizar la escritura de texto.

Ejemplo:

```
# include <stdio.h>
void main ( )
{
    puts ("Escribo una frase en pantalla");
    puts ("Escribo otra frase en la siguiente línea");
}
```

Ejecutando este ejemplo aparecen en pantalla las dos frases escritas en dos líneas consecutivas.

1.1.2 Imprimiendo datos de diferente tipo con printf

Si entre las comillas de printf aparece un símbolo %, se está indicando a la función que deje de copiar caracteres y haga algo diferente, concretamente, que imprima el contenido de una variable cuyo nombre se indica después de las comillas finales y la coma. Después del símbolo % se agrega un carácter de formateo de datos para indicar el tipo de formato en la salida.

Algunos caracteres de conversión de salida de datos son los siguientes:

d → decimal
o → octal
x → hexadecimal
c → de carácter
s → cadena
f → de punto flotante

Por ejemplo, la sentencia:

```
printf(" El valor de i es: %d \n",i);
```

escribirá el valor de la variable i en formato decimal después de la frase

El valor de i es*:* (valor)

Véase otro ejemplo:

```
#include <stdio.h>
 void main ( )
 {
      int edad=20;
      printf ("tu edad es: %x ",edad);
 }
```

Se escribirá en pantalla el valor de la variable edad en formato hexadecimal, es decir:

tu edad es: 14

Entre el signo % y el tipo de formato de salida se pueden agregar los campos siguientes de la forma %-n.m:

- -: justificación izquierda en su campo
- n: número entero que especifica la anchura mínima de campo
- .m: dígitos de la parte decimal significativa para un número en punto flotante o número mínimo de dígitos a escribir para un número entero

Ejemplo:

```
# include <stdio.h>
void main ( )
{
   float f;
   f=3.14159;
   printf ("f = % -12.5f \ n",f);
 }
```

La salida a pantalla es:

f=3.14159bbbbb (donde b indica espacio blanco)

(Véase el ejercicio 4 de las prácticas de este capítulo que incluye todas las posibilidades para los formatos de salida)

1.1.3 Agregando comentarios en C

Todo lo que aparezca escrito entre los símbolos **/*** .....***/** son comentarios para el compilador C. Los comentarios se añaden para hacer un programa más legible, pero el compilador los ignora.

Se puede insertar un comentario en cualquier parte de un programa, incluso con anterioridad a su comienzo o con posterioridad a su finalización.

No se pueden anidar comentarios.

Ejemplo:

```
#include <stdio.h>
/* Este es un comentario ignorado por el compilador*/
void main (  ) /* se puede comentar cuando quiera */
{
        printf("Comentarios en un programa "); /*se puede continuar en otra
línea*/
        printf("C \n");
}
/* más comentarios al final*/
```

1.1.4 Diseñando programas con buen estilo

Véase un ejemplo de programa bien diseñado:

```
        #include <stdio.h>

          void main (  ) /* Este programa escribe texto en pantalla */
          {
            printf("Un buen diseño ");
            printf ("puede ayudar a ");
            printf ("comprender un programa.\n");
            printf("Y un mal diseño ");
            printf ("puede hacer un programa ");
            printf ("ilegible.\n");
            }
```

La salida a pantalla es:

```
Un buen diseño puede ayudar a comprender un programa.
Un mal diseño puede hacer un programa ilegible.
```

Un ejemplo del mismo programa mal diseñado, es:

```
#include <stdio.h>
void main (  ) /* Este programa escribe texto en pantalla*/ {  printf(
"Un buen diseño ");
 printf("puede ayudar a ");printf ("comprender un programa.\n"
);printf("Y un mal diseño "); printf("puede hacer un programa "
                                                    );printf  ("ilegible.\n");}
```

Los dos programas son válidos y realizan las mismas instrucciones. La única diferencia entre ambos es el diseño.

El compilador C ignora todos los espacios extra y los saltos de línea, sin embargo, no admite un salto de línea entre las comillas de inicio y cierre de la función printf.

Nota importante:

Cuando se programa en C utilizando el compilador Visual C de Microsoft, es habitual que los resultados se visualicen en una consola del sistema operativo DOS. El editor de texto de Visual C++ guarda los archivos en formato ANSI, que es el formato preferido por los editores de texto basados en *windows*. Los editores de texto DOS generalmente utilizan el formato ASCII. Tanto el formato ANSI como el ASCII asignan un número entre 0 y 255 para cada uno de los 255 caracteres. ANSI y ASCII coinciden en la mayoría de los primeros 128 caracteres pero asignan diferentes caracteres a los números altos. Por ello, si editamos un programa con Visual C++ en el que se incluyen caracteres ANSI altos, en la salida a pantalla DOS éstos caracteres se muestran de forma distinta.
Por ejemplo, el carácter ñ tiene código ASCII 164, si queremos que aparezca como tal en DOS, se debe teclear en Visual C++ el carácter ANSI con código 164 que es ¤, o lo que es lo mismo teclear una barra inclinada seguida del número 164 en base octal, que es 244; es decir \244. Con ello la primera sentencia del programa anterior quedaría:

```
printf("Un buen dise\244o ");
```

Esta conversión debemos hacerla cuando queramos que aparezca en DOS algún carácter de código ASCII 128 a 255 (Tabla A.1) utilizando para ello la Tabla A.2.

1.2 Tipos atómicos de datos: constantes y variables. Declaración de variables.

1.2.1 Identificadores y palabras clave

Los identificadores son nombres que se les da a varios elementos de un programa, como variables, funciones y tablas.

Las reglas para crear identificadores, son:

- Se pueden usar letras y dígitos y el carácter subrayado (_)
- Debe comenzar por una letra o por subrayado, aunque esto último no es habitual.
- Se pueden utilizar mayúsculas y minúsculas (no acentuadas, en general), aunque es costumbre utilizar minúsculas para la mayoría de los compiladores.
- No se pueden intercambiar mayúsculas y minúsculas, por ejemplo, la variable A es distinta de la variable a.
- La longitud máxima de un identificador es de 31 caracteres.

Ejemplos de identificadores válidos:

```
x       y12     suma_1        _temperatura  nombres       area
```

Ejemplos de identificadores no válidos:

```
    4_num       ( el primer carácter debe ser una letra o _)
    "x"         (  caracteres ilegales (")) 
    orden-no    (  carácter ilegal ( - ))
indicador error ( carácter ilegal (blanco))
```

Hay ciertas palabras reservadas, denominadas palabras clave, que tienen en C un significado predefinido estándar. Las palabras clave sólo se pueden utilizar para su propósito ya establecido y no como identificadores creados por el programador.

Algunas palabras clave son:

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

Algunos compiladores C pueden reconocer otras palabras clave. Para obtener una lista completa de las palabras clave de un determinado compilador se debe consultar su manual de referencia.
Todas las palabras clave están en minúsculas. Recordemos que las minúsculas y mayúsculas no son equivalentes, por ello se puede utilizar una palabra clave escrita en mayúsculas como un identificador, aunque no es habitual.

1.2.2 Tipos atómicos de datos: variables y constantes

1.2.2.1 Variables

Una variable es una posición de memoria con nombre (identificador) que puede guardar distintos valores. El valor que contiene la variable puede variar a lo largo de la ejecución del programa, pero el tipo de datos asociado a la variable no puede variar.

C tiene cinco tipos diferentes de datos básicos:

int	Datos enteros con signo (4 bytes)	-2^{31} a $+2^{31}-1$
char	Dato de carácter (1 byte)	Valores de 0 a 255
float	Dato de coma flotante (real) (4 bytes)	Valores de ±3.4 E-38 a ±3.4 E+38 con 7 cifras significativas
double	Coma flotante de doble precisión (8 bytes)	Valores de ±1.7 E-308 a ±1.7 E+308 con 16 cifras significativas
void	vacío	Sin valor

En C no existe el tipo de datos lógico, es decir, verdadero o falso. Cualquier expresión cuyo valor sea 0 es identificada con el valor falso y si es distinta de 0 es identificada con el valor verdadero.

Las variables de tipo char tienen la longitud de 1 byte y pueden almacenar un único carácter, que puede ser cualquiera: una letra mayúscula o minúscula, números, signos de puntuación, etc.
El tipo char es equivalente a un entero de 1 byte y almacena enteros con signo de ï 128 a 127. Cada carácter es almacenado por su código ASCII. A una variable de tipo char podemos asignarle un valor entero (en ese intervalo) o directamente un carácter entre comillas simples. Internamente es un entero y externamente se puede mostrar como un carácter.

1.2.2.2 Declaración de variables

En C, a diferencia de otros lenguajes de programación, todas las variables se deben declarar antes de poder utilizarlas.

Una declaración asocia un tipo de datos determinado a un grupo de variables. Se deben declarar todas las variables antes de que aparezcan sentencias ejecutables. (En Visual C ++ se permite mezclar la parte declarativa y ejecutable del programa, aunque se debe cumplir la condición de que toda variable debe declararse antes de ser utilizada).

Una declaración consiste en un tipo de datos, seguido de uno o más nombres de variables finalizando con un punto y coma (;).

Ejemplos:

```
int a,b,c;
float raiz1, raiz2;
char indicador, texto;
```

También se puede disponer cada variable en una línea distinta de declaración:

```
int a;
int b;
int c;
float raiz1;
float raiz2;
```

```
char indicador;
char texto;
```

Se pueden asignar valores iniciales a las variables dentro de la declaración de tipo. Para ello, la declaración debe consistir en un tipo de datos, seguido por un nombre de variable, un signo igual (=) y una constante del tipo apropiado.

Ejemplos:

```
int c=12;
char estrella='*';
float suma=0., max=0.;
double factor=0.21023 e –6;
```

1.2.2.3 Constantes

Pueden definirse **constantes simbólicas**, que son nombres que representan una constante numérica, una constante de tipo carácter o cadena de caracteres. Se definen así:

```
#define nombre_constante valor
```

(esta sentencia no acaba en (;))

Antes del comienzo real de la compilación, el compilador resuelve todos los *define*. Buscará cada lugar del programa donde se encuentre el nombre de la constante y lo reemplazará por su valor.

Ejemplo:

```
#include <stdio.h>
#define PI 3.141593
void main ( )
{
  printf("PI = %f \ n",PI);
 }
```

Esta sentencia printf contiene dos veces la palabra PI; la primera de ellas no se verá afectada por la definición de la constante PI ya que el término "PI = %f \n" es una constante de cadena (escritura literal), la segunda hace referencia a la constante o variable PI que está definida y por tanto la sentencia equivale a:

```
printf("PI = %f\n",3.141593);
```

Las **constantes literales** son valores que aparecen directamente en el código. Pueden ser:

- **Constantes enteras**: se especifican como números sin componente fraccionaria. Por ejemplo,10 y −100 son constantes enteras.

- **Constantes de punto flotante**: requieren la utilización del punto seguido de la parte decimal del número. Por ejemplo, 11.123 es una constante en punto flotante. C también permite que se utilice notación científica para números en punto flotante de la forma: *número E signo exponente* (el signo es opcional). Por ejemplo, el valor $3x10^5$ se puede expresar, entre otras, de las siguientes formas:

300000.	3.0e+5	30.E+4
3e5	.3e6	300e3
3e+5	0.3e+6	300.0E+3
3E5	30E4	.03e7

Visual C++ considera todas las constantes en punto flotante como de tipo double, si se quiere que sea considerada como de tipo float se debe añadir al final de la constante una ñfò (mayúscula o minúscula). Ejemplo:

```
#include <stdio.h>
 void main ( )
 {
    float x=6.7F;
   printf("%f \n",x);
                }
```

- **Constantes de tipo carácter**: se encierran entre comillas simples: ά
- **Cadenas**: Una cadena es un conjunto de caracteres encerrados entre comillas dobles. Ya se ha trabajado con cadenas al estudiar la función printf. Recordemos que aunque C permite definir constantes de cadena no tiene un tipo de datos específico de cadena. En lugar de eso, como se estudiará más adelante, las cadenas se corresponden en C con vectores de caracteres.
  Para mostrar una cadena utilizando printf, se puede hacer que forme parte de la cadena de control o utilizar el código de formato **%s.**

  Ejemplo: Este programa imprime en pantalla la frase La casa es blanca

```
#include <stdio.h>
 void main (   )
 {
    printf("%s %s %s","La", "casa es", "blanca");
  }
```

En la cadena de caracteres se pueden incluir ciertas *secuencias de escape* (ciertos caracteres no imprimibles que van precedidos por \ y tienen algún significado). Se indican a continuación los más importantes:

\b → backspace (el cursor se mueve un espacio hacia atrás)
\t → tabulador horizontal
\n → nueva línea
\r → retorno de carro (el cursor pasa a la columna 1 de la línea en la que se encuentre)
\" → inserta comillas
\? →inserta signo de interrogación

Ejemplo:

```
#include <stdio.h>
 void main (  )
 {
  printf("Mi casa es \t blanca \ny la tuya \"amarilla\"");
 }
```

La salida a pantalla es:

```
Mi casa es      blanca
y la tuya "amarilla"
```

Una constante de tipo carácter (por ejemplo de valor 'A') y su correspondiente cadena de un solo carácter ("A") no son equivalentes. Una constante de carácter tiene un valor entero correspondiente, una cadena de un solo carácter no tiene un valor entero equivalente y de hecho consta de dos caracteres (\0 al final).

1.3 Operadores y Expresiones

Las constantes y variables se pueden unir con varios operadores para formar expresiones. Los operadores se pueden agrupar en diferentes categorías:

1.3.1 Operadores de asignación

- **Operador de asignación simple** (=)

Las expresiones que utilizan este operador se deben escribir de la siguiente forma:

```
L-valor = R-valor
```

Actúa evaluando en primer lugar la expresión de la derecha, y el valor que se obtiene se almacena en el L-valor de la izquierda.

Un L-valor puede ser una variable escalar, la componente de una tabla, etc.

Un R-valor puede ser una operación entre variables o entre constantes y variables.
En general un L-valor denota una dirección de almacenamiento en la memoria, mientras que un R-valor es una expresión que tras ser evaluada produce una constante que es almacenada en la expresión de la izquierda. Si lo que aparece a la derecha del signo igual es un L-valor, entonces, la expresión que se asigna es el contenido de la zona de memoria definida por el L-valor.

Ejemplos:

Sería válido

```
a=x+y
```

pero no lo sería

```
x + y = a,
```

pues lo que aparece a la izquierda es un R-valor.

También se pueden producir asignaciones múltiples de la forma `i=j=5,` de esta forma, el valor 5 se asigna a j y el valor j se asigna a i.

- **Operadores de asignación compuestos (+ =, - =, * =, / =, % =)**

Para estudiar su utilización consideremos el primer operador `+ =`

```
Variable += expresión
```

es equivalente a

```
Variable = Variable + expresión
```

También se pueden encontrar asignaciones compuestas múltiples de la forma:

```
i+=j+=k;
```

que será equivalente a las dos sentencias siguientes:

```
j=j+k;
i=i+j;
```

Todos los operadores de asignación tienen la misma prioridad. En el caso de encontrarse varios de ellos seguidos en la misma expresión se realizarán de derecha a izquierda.

En las asignaciones el tipo de dato de la expresión (R-valor) se convierte al de la variable (L-valor) a la que se asigna. Ejemplo:

```
 include <stdio.h>
void main (  )
{
   int f;
   f=31.678; /*  si se asigna un valor real
             a un dato entero, en general, toma la
             parte entera del dato, aunque avisa (warning)
             que puede ocurrir una pérdida de datos */

   printf("f=%12d\n",f);
}
```

La salida es:

```
f=          31
```

1.3.2 Operadores aritméticos

Actúan entre operandos numéricos: enteros, de coma flotante o carácter. Son los siguientes:

+	Suma o signo positivo
-	Resta o signo negativo
*****	Multiplicación
/	División
%	Módulo (resto de la división entera)

El operador resto (%) requiere que los dos operandos sean enteros; el operador división (/) requiere que el segundo operando sea no nulo.

No hay operador de potenciación en C. Sin embargo hay una función de biblioteca ï **pow(a,b**)- que es de tipo double y pertenece al archivo de cabecera math.h, que realiza la potenciación (*a* elevado a la potencia *b*). Por ejemplo,

```
#include <stdio.h>
#include <math.h>
void main ( )
{
  double d1=3.,d2=2.,d3;
   d3=pow(d1,d2);
   printf("%f",d3);
  }
```

en el que la variable d3 toma el valor $d1^{d2}$

Orden de prioridad de los operadores aritméticos

*** / %**	+ preferencia
+ -	- preferencia

A igual preferencia, las operaciones se realizan de izquierda a derecha (en el orden que estén en la expresión), por ejemplo:

a – b / c * d es equivalente a (a – ((b / c) * d))

El resultado de una operación en la que los operandos sean del mismo tipo es obligatoriamente de ese tipo:

1 + 3 da como resultado 4
1. + 3. da como resultado 4.
4/3 da como resultado 1
4./3. da como resultado 1.333

Si los operandos difieren en el tipo pueden sufrir una conversión de tipo antes de que la expresión alcance su valor final. En general, el resultado final se expresará con la mayor precisión posible, de forma consistente con los tipos de

datos de los operandos. Se han plasmado todas las posibilidades en la siguiente tabla:

OPERANDOS	CONVERSIÓN	RESULTADO FINAL
coma flotante con distinta precisión float y double	El operando de menor precisión (float) se transforma a la precisión del otro (double)	double
coma flotante y char o int	char o int pasan al tipo coma flotante	coma flotante

Ejemplos:

- Sean C1 y C2 variables tipo char, con valores respectivos 'P' (código 80) y 'T' (código 84). Las siguientes operaciones dan como resultado:
  C1 + C2 → 164
  C1 + C2 + 5 → 169
  C1 + C2 + '5' → 217 (el carácter '5' tiene código 53)

- Sean las variables: *i* entera con valor 7; *f* tipo double con valor 5.5; *c* tipo char con valor 'w' (código 119). El valor y tipo del resultado de las siguientes operaciones es:
  i + f → 12.5 (double)
  i + c → 126 (entero)
  i + c – '0' → 78 (entero) ('0' tiene código 48)
  (i + c) – (2 * f / 5) → 123.8 (double)

Se puede convertir el valor resultante de una expresión a un tipo de datos diferente; para ello, la expresión debe ir precedida por el nombre del tipo de datos deseado entre paréntesis, de la forma: (tipo datos) expresión

Ejemplo:

- Sea la variable *i* entera de valor 7 y *f* de coma flotante de valor 8.5. La expresión `(i+f) % 4` es errónea ya que `(i+f)` es de tipo coma flotante y no puede aplicársele el operador resto. Para que fuera válida se debería incluir: `((int) (i+f)) % 4`

1.3.3 Operadores monarios

Actúan sobre un solo operando para producir un nuevo valor. Se escriben algunos delante y otros detrás del operando. Son los siguientes:

- **Operador menos** (-): cambia el signo a la constante o expresión que vaya detrás. Se puede aplicar a los tipos de datos numéricos y `char`.

- **Operador incremento** (++): incrementa en 1 al operando.
  Operador decremento (--): decrementa en 1 al operando.
  Se pueden utilizar de dos formas:
  1) Si el operador precede al operando `(++ i)` el valor del operando se modifica antes de que se utilice para cualquier otro propósito.
  2) Si el operador sigue al operando `(i ++)` el valor del operando se modificará después de ser utilizado.

Ejemplo:

```
#include <stdio.h>
void main ( )
{
   int i=1;
   printf("i = %d \n", i);
   printf("i = %d \n", ++i);
   printf("i = %d \n", i);
}
```

La salida a pantalla es:

```
i = 1
i = 2
i = 2
```

A continuación, se realiza el mismo ejemplo cambiando el operador monario de lugar:

```
#include <stdio.h>
 void main (  )
 {
      int i=1;
      printf("i = %d\n", i);
      printf("i = %d\n", i++);
      printf("i = %d\n", i);
 }
```

La salida a pantalla es:

```
i = 1
i = 1
i = 2
```

Otro ejemplo algo más complejo:

```
#include <stdio.h>
void main ( )
{
        int y=2, z=1025;
        z= y++; /* y vale 2, le paso ese valor a z y después incremento
y */
        printf("z e y valen %d %d\n", z,y);
        z=++y; /* y vale 3, incremento su valor (4) y se lo paso a z*/
        printf("z e y valen %d %d\n", z,y);
        y=3;
        z=y--; /* y vale 3, le paso ese valor a z y después decremento y
*/
        printf("z e y valen %d %d\n", z,y);
        z=--y;/* y vale 2, decremento su valor (1) y se lo paso a z --
>1*/
```

```
        printf("z e y valen %d %d\n", z,y);
}
```

La salida a pantalla es:

z e y valen 2 3
z e y valen 4 4
z e y valen 3 2
z e y valen 1 1

- **Operador sizeof( )**. Este operador devuelve el tamaño de su operando en bytes. La información obtenida puede ser muy útil cuando se transfiere el programa a una computadora diferente o a otro compilador C. También se utiliza para la asignación dinámica de memoria. Por ejemplo, sea i una variable tipo int, la sentencia

  printf ("int: %d\n", **sizeof**(i));

  da como salida en pantalla: int: 4

En general, un operador monario tiene más prioridad que uno aritmético. A igual prioridad se realizará la operación de derecha a izquierda.

Sin embargo, esta regla general de prioridad no es aplicable a los operadores incremento y decremento que tendrán más o menos prioridad que todos los operadores de la sentencia en la que se ubiquen dependiendo de si se encuentran antes o después de la variable sobre la que actúen. Veamos el siguiente ejemplo

```
int x=2, y=3,z;
z=(x++ +y)*x;
```

El resultado es z = 10, ya que se calcula todo el R-valor y posteriormente se incrementa x, que pasará a tener valor 3 cuando se use a continuación.

1.3.4 Operadores relacionales y de igualdad

Actúan sobre dos operandos numéricos o de carácter y dan como resultado un valor lógico (falso o verdadero) que realmente se almacena como un entero (0 falso, 1 verdadero). Son los siguientes:

RELACIONALES	
<	Menor que
< =	Menor o igual que
>	Mayor que
> =	Mayor o igual que
IGUALDAD	
= =	Igual a
! =	Distinto de

Los operadores relacionales tienen mayor prioridad que los de igualdad. Entre ellos tienen la misma prioridad que se resuelve de izquierda a derecha.

Los operadores aritméticos tienen mayor prioridad que éstos.

Ejemplos:

```
int i=1, j=2, k=3;
```

Expresión	Interpretación	Valor
i < j	Cierto	1
1 + j >= k	Cierto	1
j + k > i + 5	Falso	0
k ! = 3	Falso	0
j = = 2	Cierto	1

Cuando se realizan comparaciones, si los operandos son de diferente tipo se convertirán al tipo más amplio de los dos.

```
int i=7;
float f= 5.5;
```

```
char c = 'w';  (código 119)
```

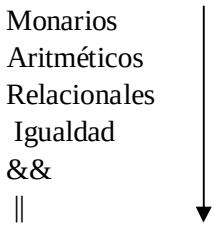
Expresión	Interpretación	Valor
f > 5	Cierto	1
i + f <= 10	Falso	0
c = = 119	Cierto	1
c ! = 'p'	Cierto	1
c > = 10 * (i + f)	Falso	0

1.3.5 Operadores lógicos

Actúan sobre operandos lógicos dando como resultado un valor lógico. Son los siguientes:

- **&&** (Y lógico ï AND). Actúa sobre dos operandos lógicos y el resultado es verdadero (1) sólo cuando los dos operandos valen verdadero.
- **||** (O lógico ï OR). Actúa sobre dos operandos lógicos y el resultado es verdadero (1) cuando uno o los dos operandos valen verdadero.
- ! (negación lógica). Actúa sobre un único operando a su derecha y lo cambia de valor. Es realmente un operador monario y tiene la misma prioridad que éstos.

El orden de prioridad de los operadores vistos hasta ahora (de mayor a menor):



Ejemplo:

```
#include <stdio.h>
 void main ()
{
 int i=7;
 float f=5.5;
 char c='w';
 printf ("lógico: %d\n", f> 5);
 printf ("lógico: %d\n", !(f> 5));
 printf ("lógico: %d\n", i <= 3);
 printf ("lógico: %d\n", !(i<=3));
 printf ("lógico: %d\n", i> f +1);
 printf ("lógico: %d\n", !(i>f+1));
 printf ("lógico: %d\n", i>=6&&c=='w');
 printf ("lógico: %d\n", c!='p'||i+f<=10);
}
```

La salida a pantalla es:

```
lógico: 1
lógico: 0
lógico: 0
lógico: 1
lógico: 1
lógico: 0
lógico: 1
lógico: 1
```

1.3.6 Operador condicional

Se pueden hacer operaciones condicionales simples con el operador condicional **(**? **:)**. Una expresión que hace uso del operador condicional se

denomina expresión condicional. Se puede utilizar una expresión condicional en lugar de la sentencia más tradicional if-else que se estudiará en el capítulo 4.

Sintaxis:

```
expresión_1?expresión_2:expresión_3
```

La expresión condicional actúa de la siguiente forma:

- Si la expresión_1 es cierta, entonces se evalúa la expresión_2 y ese valor es el que toma la expresión condicional.
- Si la expresión_1 es falsa, entonces se evalúa la expresión_3 y ese valor es el que toma la expresión condicional

Ejemplo:

```
#include <stdio.h>
 void main ( )
{
     int i=7;
   printf ("valor de la expresion condicional: %d \n", i<0 ? 0:100);
}
```

En este caso, i<0 es falso luego la expresión toma el valor 100.

Este operador tiene menor prioridad que el O lógico (||) y mayor que los operadores de asignación.

ORDEN DE PRIORIDAD DE LOS OPERADORES (de arriba abajo)

		A igual prioridad:
Paréntesis	( )	
Monarios	! , - , ++, --, sizeof	Derecha a izquierda
Multiplicación, división y resto	* , / , %	Izquierda a derecha
Suma y resta	+ , -	Izquierda a derecha
Relacionales	< , < = , > , > =	Izquierda a derecha
Igualdad	= =, ! =	Izquierda a derecha
Y lógico	&&	Izquierda a derecha
O lógico	||	Izquierda a derecha
Condicional	? :	Derecha a izquierda
Asignación	= , +=, -=, *=, /=, %=	Derecha a izquierda

1.4 Entrada de datos por teclado

Se pueden introducir datos en el ordenador procedentes del teclado con la función de biblioteca **scanf**. Se encuentra en la librería <stdio.h>. Esta función se puede utilizar para introducir cualquier combinación de valores numéricos, caracteres sueltos y cadenas de caracteres.

Sintaxis:

```
scanf(" ...% formato_1 .... % formato_2  .....% formato_N",
& var1, &var2, ... , &varN)
```

Los datos introducidos por teclado se leerán según el formato_i, y se almacenarán en memoria con el nombre var_i.

Especificadores de formato para la entrada de datos:

%c ------ lee un solo carácter
%d ------ entero decimal
%e ------ coma flotante

%f ------ coma flotante simple precisión
%lf ------ coma flotante doble precisión
%g ------ coma flotante
%i ------ entero decimal, octal o hexadecimal
%o ------ entero octal
%s ------ cadena de caracteres seguida de un espacio
%x ------ entero hexadecimal

Los tipos de las variables deben coincidir uno a uno con los indicados en el formato. Cada nombre de la variable debe ir precedido por un ampersand (&). (En realidad, los argumentos son punteros que indican dónde se encuentran situados los datos en memoria (dirección de memoria), tal como se estudiará en el capítulo 6.) Los nombres de arrays no deben ser precedidos por el ampersand.

Ejemplo:

```
#include <stdio.h>
void main ( )
{
     int entero;
     char letras[20];
   double coste;
 scanf("%d %s %lf",&entero, letras, &coste);
     printf("%d %s %f",entero, letras, coste);
}
```

Los datos se introducen separados por espacios o saltos de línea. Por ejemplo se podría haber introducido:

123 velocidad 8.7

Se memorizará el valor 123 en la variable entero, el valor 8.7 en la variable coste, y los caracteres *velocidad* en los 9 primeros componentes de letras. (letras es un array, se explicará con detalle en el próximo capítulo).

Se va a estudiar con mayor detalle la **lectura de cadenas de caracteres** por su mayor complejidad respecto a la lectura de otros tipos de datos:

El formato %s se aplica a cadenas de caracteres que acaban en un carácter de espaciado. Es decir, con este formato se leerá la cadena introducida hasta el primer blanco. Por tanto, una cadena que incluye caracteres de espaciado intermedio no se puede introducir de esta forma ya que la lectura terminaría en el primer espacio.

Para trabajar con cadenas que incluyen caracteres de espaciado:

- Se puede utilizar la función getchar que funciona leyendo un único carácter por teclado. La función se introducirá dentro de un bucle que repita la lectura el número de veces necesario. Se estudiará con detalle en el próximo capítulo.
- También se puede emplear la función getche, que funciona de forma semejante a la anterior.
- Se puede utilizar la función scanf, incluyendo en el formato después de % y entre corchetes los caracteres que se van a admitir para la cadena de lectura. (La cadena de caracteres terminará cuando encuentre un carácter de entrada que no coincida con ninguno de los que están entre corchetes; se añadirá entonces un carácter nulo al final de la cadena (\0)

Ejemplo:

```
#include <stdio.h>
void main ( )
{
   char linea[80];
        scanf("%[
ABCDEFGHIJKLMNOPQRSTUVWXYZ]",linea);
        printf("%s ", linea);
}
```

Si se introduce la cadena:

LA CASA BLANCA

se le asignará al array *linea* toda la cadena de caracteres, ya que está compuesta por mayúsculas y espacios en blanco. Sin embargo si se hubiera introducido la cadena

La casa blanca

sólo se habría asignado el carácter L a *linea*, ya que la segunda letra (a minúscula) no es un carácter válido y se interpreta como el primer carácter a continuación de la cadena. Se podrían haber incluido las minúsculas dentro de los corchetes, pero sería muy engorroso. Para evitarlo, se pueden preceder los caracteres de dentro del corchete por el símbolo ^ que significa que se leerían todos los caracteres que no coincidan con los indicados en los corchetes.

Mediante la sentencia:

```
scanf ("%[^\n]", linea);
```

no habrá restricciones en los caracteres que compongan la cadena, salvo que quepan en una línea (se leen todos los caracteres menos el salto de línea, el cual finaliza la lectura).

- Se puede emplear la función gets que lee la cadena hasta el primer salto de línea y la introduce en la variable que tiene como argumento. Ejemplo:

```
gets(cad);/* el texto leído se introduce en la variable cad*/
```

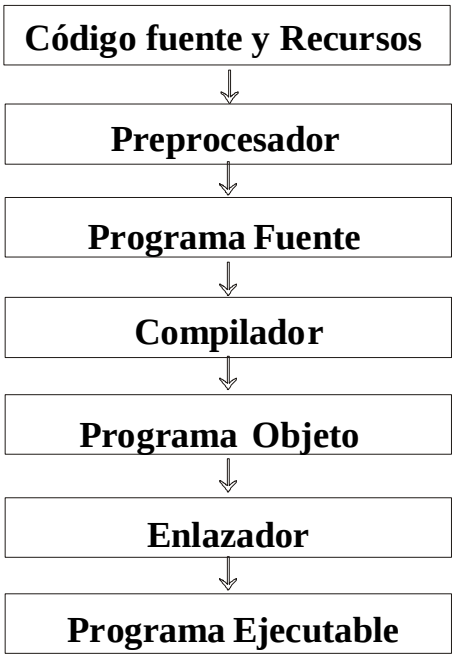
1.5 Enunciados de las prácticas

Ejercicio 1. Creación de una Aplicación en C/C++ usando el Developer Studio de Microsoft

Este ejercicio tiene un formato distinto a los posteriores ya que no exige la resolución de un problema sino que, únicamente, ofrece los pasos a seguir para crear un programa C usando el Developer Studio de Microsoft. Es necesario el seguimiento de este ejercicio para la toma de contacto con un compilador que permita la ejecución de los programas que se realizarán a lo largo del curso, por ello se propone como primera práctica del mismo.

1.1. Esquema General del Desarrollo de una Aplicación en C/C++

Para crear un programa que sea ñentendidoò y pueda ser ejecutado por el ordenador, es necesario que el programador escriba el código fuente en un lenguaje de alto nivel (en este caso C/C++) y tras añadir los recursos (gráficos, etc) que necesita el código se produzca una ñtraducciónò a código binario de acuerdo con el siguiente esquema:



El **preprocesador** es un programa que realiza las siguientes acciones sobre el código fuente:

- Incluye archivos y archivos de cabecera (*.h*), indicados en sentencias de tipo *#include*.
- Sustituye macros, indicados por sentencias de tipo *#define*.

El **compilador** actúa a continuación y transforma el código generado por el preprocesador en código binario (extensión *.obj*).

Por último actúa el **enlazador** para incluir el código necesario de la biblioteca de funciones del sistema. El enlazador genera el programa ejecutable (extensión *.exe*).

1.2. El Entorno de Desarrollo Developer Studio de Microsoft

El paquete Visual C++ incluye muchas piezas separadas: editores (programas que permiten escribir ficheros), compiladores, enlazadores y otras herramientas que permiten desarrollar programas C/C++ para Microsoft Windows. El entorno de desarrollo Developer Studio reúne todas las herramientas de Visual C++ en un entorno integrado.

En ciertas aplicaciones profesionales el Developer Studio permite utilizar la Microsoft Foundation Class Library (MFC) que es un conjunto de clases C++ que comprenden una aplicación genérica de funcionamiento de Windows lista para que el usuario la personalice.

En este libro, orientado a las aplicaciones numéricas de la Ingeniería, solamente se va a utilizar el Developer Studio para el desarrollo de programas en C.

La unidad básica de desarrollo en el Developer Studio se llama **PROYECTO**.

Cuando se empieza un programa nuevo, lo primero que hay que hacer es generar un proyecto para el mismo. Un proyecto es un conjunto de archivos, que

están situados en la carpeta que indique el usuario en la definición de éste, y que son tratados conjuntamente por el Developer Studio para crear un programa ejecutable. Se suele dar el mismo nombre a la carpeta de proyecto que al proyecto, aunque no es estrictamente necesario.

Dentro de una carpeta de proyecto se encuentran los siguientes tipos de archivos fuente:

1) *Archivos de Cabecera* : son archivos *include* que tienen extensión *.h*

2) *Archivos de Implementación o Código Fuente* : tienen la extensión *.cpp* para C++ o la extensión *.c* para C.

3) *Archivos de Recurso* : tienen la extensión *.rc*. Además pueden aparecer archivos con extensiones *.bmp* o *.ico* como imágenes de botones (bitmaps) o iconos.

Aparte de los archivos fuente, aparecen un conjunto de archivos auxiliares, un archivo de proyecto de extensión *.dsp* y un archivo espacio de trabajo de extensión *.dsw*.

Dentro de la carpeta PROYECTO aparecen , en general, dos subcarpetas:

1) *Carpeta Debug* : en esta carpeta se crean e incluyen los ficheros intermedios de código objeto (*.obj* y otros), y se incluye el fichero ejecutable del proyecto (*.exe*).

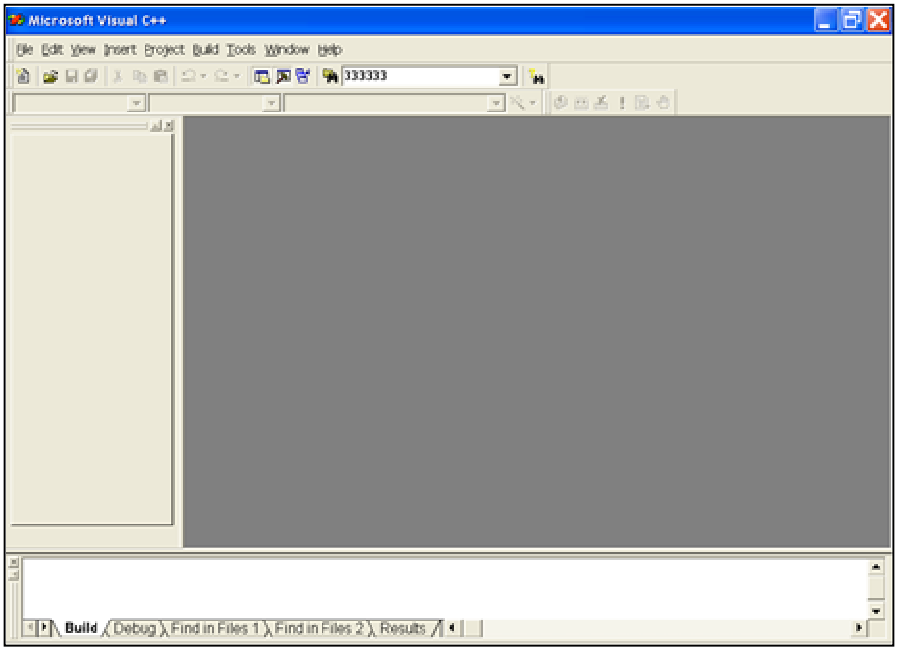
2) *Carpeta Res* : en esta carpeta se incluyen los archivos de Recursos.

Cuando se genera un proyecto, también se crea por defecto un espacio de trabajo (workspace); su archivo (citado arriba), de extensión .dsw y nombre el del proyecto, guarda información sobre las ventanas de Visual C++ que están abiertas así como cualquier otro ajuste especificado.

1.3. Utilización del Developer Studio

Se va a proceder a crear un proyecto de nombre *primero*. El alumno debe seguir los pasos indicados a continuación:

1. Desde ñmi PCò o ñexplorador de windowsò, crear una carpeta con el nombre del alumno, por ejemplo ñJ.Sanzò. Esta carpeta contendrá de ahora en adelante, todos los proyectos C que cada estudiante va a realizar a lo largo del curso.
2. Iniciar el Visual C++; aparece el siguiente entorno integrado:



3. Desplegar el menú **File** y elegir la opción *New*.
4. En el cuadro de diálogo **New** y en la pestaña *Projects*, elegir *Win32 Console Application*.
5. Escribir **primero** en el cuadro *Project Name*.
6. En *Location*, escribir la dirección del Proyecto: C:\J.Sanz\primero
7. Seleccionar la opción *Create new workspace*.
8. Hacer clic en el botón OK.
9. Aparece el cuadro de diálogo: *Win 32 Console Application Step 1 of 1*.
10. Elegir *An empty Project* y hacer clic en el botón *Finish*.
11. Aparece un cuadro de diálogo con información acerca del nuevo proyecto.

1.4. Mi primer programa en C

A continuación, se va a generar el fichero que contiene el código fuente asociado al proyecto creado en el punto anterior:

1. Desplegar el menú **File** y elegir la opción *New*.
2. De la pestaña Files elegir *C++ Source File*.
3. Comprobar que está señalada la opción *Add to project: ... primero.*
4. En el cuadro de diálogo *File name* escribir **hola.c** y hacer clic en el botón OK. Aparece el editor de texto de Visual C++.
5. En el editor de texto escribir el siguiente programa en C:

```
# include <stdio.h>
    void  main(  )
    {
            printf("Hola Mundo \n");
    }
```

6. Desplegar el menú **Build** y elegir la opción *Build primero.exe*. Si no hay errores se produce la compilación y el enlace con librerias o (linkado) de **hola.c**. También se puede seleccionar esta opción pulsando el botón

7. Desplegar el menú **Build** y elegir la opción *Execute primero.exe*. También se puede seleccionar esta opción pulsando el botón

Entonces se abre una ventana de DOS y aparece el mensaje ñHola Mundoò.
Si pulsamos la tecla Intro, se cierra la ventana con la salida del resultado.

8. A continuación ir al menú **File** y elegir la opción *Exit*. De esta manera se cierra el Developer Studio.

1.5. Análisis del Proceso Creado

Abrir desde Windows la carpeta ñprimeroò situada dentro de la carpeta ñJ.Sanzò. Aparecen los siguientes ficheros:

hola.c (archivo de código fuente).
primero.opt, primero.ncb y primero.html (archivos auxiliares).

primero.dsp (archivo de proyecto).
primero.dsw (espacio de trabajo del proyecto).

y la carpeta: **Debug**

Esta carpeta contiene varios archivos intermedios correspondientes al proceso de compilación y enlazado y un fichero ejecutable llamado ñprimero.exeò.

1.6. Ejecución posterior de un proyecto ya creado.

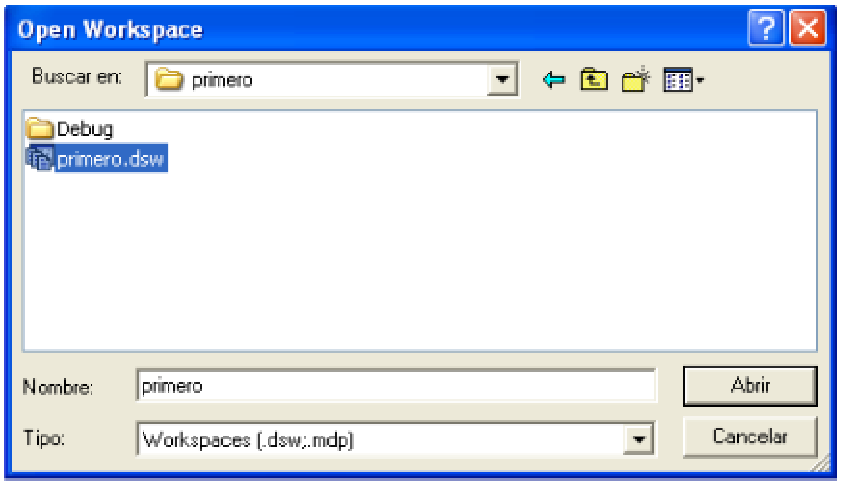
Para ejecutar un programa que ha sido creado siguiendo el proceso anterior y que se encuentra grabado en el computador, se seguirán los pasos indicados a continuación:

1. Abrir el Developer Studio.
2. Desplegar el menú **File** y elegir la opción *Open*. En el cuadro de diálogo Abrir, elegir C:\J.Sanz\primero\Debug
3. Desplegar el cuadro combinado y elegir la opción *Executable Files (.exe)*. Seleccionar ***ñ****primero****ò*** y *Abrir*.
9. 4. Desplegar el menú **Build** y seleccionar la opción *! Execute primero.exe*. El programa se ejecuta. Pulsar intro y cerrar.
4. Salir y no guardar

1.7. Apertura de un proyecto ya creado para su modificación.

Para trabajar con un proyecto ya existente se debe abrir el espacio de trabajo o archivo de extensión *.dsw*. Con los siguientes pasos vamos a abrir el proyecto *primero*, cambiar código del fichero fuente y volver a compilar y ejecutar:

1. Abrir el Developer Studio.
2. Desplegar el menú **File** y elegir la opción Open Workspace. Aparece un cuadro de diálogo que hay que completar como se muestra a continuación y *Abrir*:



Al abrir el espacio de trabajo aparece Visual C++ con la apariencia que tenía la última vez que se guardó ese documento (las mismas ventanas con la misma información).

3. En el código fuente, cambiar la frase ñHola Mundoò por ñHola de nuevo, Mundoò.
4. Volvemos a compilar mediante el menú **Build** y la opción *Build primero.exe*
5. Ejecutar con la opción *¡Execute primero.exe*
6. Guardar todo.

1.8. Corrección de errores en un programa

Veamos cómo se puede corregir un programa cuando se detectan errores en la compilación:

1. Siguiendo los pasos indicados en los apartados 1.3 y 1.4, crear un nuevo proyecto de nombre **segundo** dentro de la carpeta ñJ.Sanzò. Dentro de este proyecto incluir un programa en C (C++ Source File) de nombre **holaerr.c** con el siguiente código:

```
# include <stdio.h>
     void  main(  )
{
    printf("Hola Mundo \n")
}
```

2. Desplegar el menu **Build** y elegir la opción *Build segundo.exe.* En la parte inferior de la pantalla y dentro de la *Output Window* aparece el mensaje:

   c:\j.sanz\segundo\holaerr.c(5)
   Error C2143: syntax error: missing **á;ô** *before* **ó;â**
   Segundo.exe- 1 error(s), 0 warning(s)

   Si se hace doble clic en la línea de la ventana *Output Window* donde se describe el error, entonces queda marcada la línea de código fuente donde el error está situado.

3. Corregir el error colocando ñ;ò detrás de la sentencia que comienza por printf o antes de la llave de cierre.

4. Volver a desplegar el menú **Build** y elegir la opción *Build segundo.exe.*

5. En la Output Window ahora aparece: 0 error(s), 0 warning(s). Se ha generado el código ejecutable.

6. Ahora lo ejecutamos haciendo clic en el icono ! que está debajo de la barra de menús.

1.9. Creación de un Fichero de Texto para incluir datos. Manejo del editor de Visual C++.

Tal como se estudiará en el capítulo 8, los datos suministrados a un programa pueden insertarse por teclado o estar almacenados en un fichero. Visual C++ permite crear un fichero de texto de datos en su propio entorno de forma que se pueda manejar de manera sencilla por el usuario:

1. Desplegar el Menu **File**. Aparece la opción *New*.
2. De la pestaña **File** seleccionar la opción *Text File*.
3. En el cuadro *Location* seleccionar: c:\J.Sanz\primero.
4. En el cuadro **File Name** escribir *datos*.
5. Aparece un editor de texto, escribir en él las siguientes tres filas de datos:

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

6. Utilizando el cursor del ratón y manteniendo presionado el botón izquierdo, marcar el anterior grupo de datos. Hacer clic en el icono ***Copy*** situado debajo de la barra de menús.
   Con la tecla intro y la barra espaciadora llevar el cursor a otra posición de la pantalla y duplicar los datos haciendo clic en el icono ***Paste***. Repetir este proceso unas cuantas veces. Seleccionar ahora un grupo numérico y borrarlo utilizando la tecla Supr.

7. La BÚSQUEDA de un fragmento de texto en un fichero se puede hacer de dos maneras distintas:

   -Secuencial: Supongamos que se quiere saber dónde aparece el grupo 2 2 2 2 2. En el menú **Edit** seleccionamos *Find* y en el cuadro de diálogo que aparece escribimos 2 2 2 2 2 en el cuadro de texto *Find what;* a continuación se hace clic en *Find Next*. Repitiendo esta operación se puede encontrar de forma sucesiva, las veces que aparece 2 2 2 2 2 en el texto.

   -Global: Si se desean señalizar a la vez todos los lugares donde aparece el grupo 1 1 1 1 1 , seleccionamos la opción *Find* después de desplegar el menú **Edit**. Aparece el cuadro de diálogo y escribimos 1 1 1 1 1 dentro del cuadro de texto *Find what*. A continuación se hace clic en *Mark All* y quedan señalizados todos los lugares donde aparece 1 1 1 1 1.

8. Se puede REEMPLAZAR un determinado fragmento de texto por otro cualquiera, del siguiente modo:

   Supongamos que se desea cambiar la cadena de caracteres 3 3 3 3 3 por AAA en todo el texto. Para ello se despliega el menú **Edit** y dentro se selecciona la opción *Replace*. En el cuadro *Find what* escribimos 3 3 3 3 3, en el cuadro *Replace with* escribimos AAA y hacemos clic en el botón Replace All.

9. Para guardar el fichero de texto, se despliega el menú **File** y se selecciona *Save*. El fichero queda guardado con el nombre **datos.txt**. (Para guardarlo con otro nombre utilizar la opción *Save As* del menú **File**).

SALIDA A PANTALLA

Ejercicio 2. Escritura de texto

Diseñar un programa C que produzca la siguiente salida a pantalla:

Este es mi primer programa C.
Programar es sencillo y divertido.

Escribir el programa de las cuatro formas siguientes:

a) Utilizando la función printf una vez
b) Utilizando la función printf dos veces
c) Utilizando la función puts una vez
d) Utilizando la función puts dos veces

Incluir en el programa un comentario que diga: Instrucción para escribir en pantalla

Ejercicio 3. Escritura de texto y datos

La función printf(…) permite escribir valores de diferente tipo simultáneamente.

Diseñar un programa C que declare dos variables enteras de nombres n1 y n2 y valores respectivos 10 y 3 y que utilizando estas variables obtenga como salida a pantalla:

El numero 10 es mayor que el numero 3.

Ejercicio 4. Escritura con formato

Determinar la salida a pantalla del siguiente programa C:

```
# include < stdio.h>

void main ( )
{
        int n=7;
       float f;
        f=3.14159;
        printf("f=%12.4f\n",f);
        printf("f=%12.3f\n",f);
        printf("f=%-12.4f\n",f);
        printf("f=%12.5f\n",f);
        printf("f=%12.6f\n",f);
        printf("f=%7f\n",f);
        printf("f=%6f\n",f);
        printf("f=%.4f\n",f);
        printf("n=%.4d\n",n);
        printf("n=%7.4d\n",n);
}
```

TIPOS DE DATOS

Ejercicio 5. Trabajando con datos tipo carácter

En C un carácter es equivalente a un número entero, ya que se memoriza según un entero que es su código ASCII.

Determinar la salida a pantalla del siguiente programa C, sabiendo que el carácter ôMô tiene código 77:

```
/*
 * En este ejemplo se imprimen en pantalla dos caracteres utilizando
 * el especificador %c y también el especificador %d, es decir, un
 * carácter se puede ver como un número entero, que es  su código ASCII
 */
```

```
#include<stdio.h>
 void main( )
  {
     char caracter;
     caracter = 'M';

     /* Imprime el carácter como carácter o como número */
     printf("El caracter %c como letra es %c\n", caracter, caracter);
     printf("El caracter %c como numero es %d\n", caracter, caracter);
     printf("Podemos operar con la letra\n");
     printf("La letra + 1 vale= %d\n",caracter+1);
     caracter=caracter+1;
     printf("La letra + 1 como caracter vale= %c\n",caracter);

  }
```

Ejercicio 6. Otro ejercicio similar

Determinar la salida a pantalla del siguiente programa C sabiendo que el carácter óAô tiene código 65 y el carácter ó0ô tiene código 48:

```
#include <stdio.h>
 void main( )

  {
     char car;
     int codigo;
     car = 'A';
     codigo=48;

     printf("El codigo ASCII para la letra %c es %d\n", car, car);
     printf("El codigo ASCII %d representa al caracter %c \n", codigo, codigo);
  }
```

Ejercicio 7. Conversión de datos

En C, como en otros lenguajes, se admite que en determinadas operaciones los operandos sean variables de tipos diferentes, aunque es conveniente saber lo que ocurre en esas circunstancias.

Determinar la salida a pantalla del siguiente programa C:

```
/*
 * En este programa se observan las distintas conversiones de unos tipos de
 * datos en otros, de acuerdo con la variable a la que son asignados.
 */

#include<stdio.h>
 void main( )
 {
    char cv;
    int i1 =10.3,i2=3 ;
    float x1, x2;

   printf("\n Perdida de decimales. i1 solo almacena enteros. i1=%d",i1);
   printf("\n El resultado de dividir %d entre %d es %d",i1,i2,i1/i2);
   x1=i1;
   printf("\n Al asignar a x1 el valor de i1, x1 queda %f",x1);
   printf("\n Podemos sumar un entero y un real:x1+i1=%f",x1+i1);
   cv=i2;
   printf("\n Entero asignado a un carácter: %d → %d (%c)\n", i2, cv, cv);
   x2=i2/2;
  printf ("aritmética entera: %d / 2 = %f \n", i2, x2);
  x2=i2/2.;
  printf ("aritmética en punto flotante: %d / 2. = %f \n", i2, x2);
 }
```

OPERADORES

Ejercicio 8. Operadores aritméticos y asignaciones

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>
void main( )
{
      int i,j=3,k,m;
      float a,b=2.,c,d;
      a=2/3*3;
      i=2./3*3;
      b+=a/j;
```

```
      k=5.4+b*2;
      d=i+k/i*i;
      m=i+k/d*d;
      c=1/2+1/2;
      printf("\n a=%f b=%f c=%f d=%f ",a,b,c,d);
      printf("\n i=%d j=%d k=%d m=%d ",i,j,k,m);
}
```

Ejercicio 9. Operador monario sizeof ( )

El siguiente programa pregunta al compilador el "tamaño de" (size of) cada uno de los distintos tipos de datos. Compilar el siguiente programa y anotar el resultado, de cara a la declaración de variables que necesitamos en cada programa.

```
/*
  * Este programa muestra el número de bytes que
  * se reserva para cada uno de los tipos de datos
  * especificados
  */
  #include <stdio.h>

  void main( )
   {
     printf("Tipo \t Bytes\n");
     printf("char \t %d\n", sizeof (char));
     printf("int \t %d\n", sizeof (int));
     printf("float \t %d\n", sizeof (float));
     printf("double \t %d\n", sizeof (double));
   }
```

Ejercicio 10. Operadores monarios incremento y decremento

Con este ejemplo se estudia como actúan los operadores de incremento y decremento, según aparezcan a la izquierda o derecha de la variable.

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>
void main( )
 {
   int x = 2, y = 3, resultado;

   printf("VALORES INICIALES: x = %d; y = %d\n", x, y);
   resultado = ++x + y;
   printf("PREFIJO: El resultado de ++x + y = %d \n", resultado);
   printf("(Despues del calculo, x = %d y y = %d)\n", x, y);

   x = 2, y = 3;    /* Restaura los valores */
   printf("VALORES INICIALES: x = %d; y = %d\n", x, y);
   resultado = x++ + y;
   printf("SUFIJO: El resultado de x++ + y = %d\n", resultado);
   printf("(Despues del calculo, x = %d y y = %d)\n", x, y);
 }
```

Ejercicio 11. Operadores de asignación, relacionales y de igualdad

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>
void main( )
{
   int a,x;
   printf("%d\n",1!=2);
   printf("%d\n",(a=2*2)= =2);
   printf("%d\n",a);
   printf("%d\n",(x=1+1)= =2);
   printf("%d\n", a<x);
   printf("%d\n",x=5);
   printf("%d\n", x>=a);
}
```

Ejercicio 12. Expresiones lógicas

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>
void main( )
{
   float a=3.,b=5.;    int c,d,e,f;
   c=a<5.;    d=a>=b;
   e=b<=6.;
   f=b == a;
   printf("%d %d %d %d\n",c,d,e,f);
   printf("%d\n", c||d && !e = = f );
   printf("%d\n", (c||d && !e) = = f );
}
```

Ejercicio 13. Expresiones lógicas (2)

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>
void main( )
{
   int a=0,b=1,c=0;
   printf("%d \n",!a || b);
   printf("%d\n", a == (b && c) );
   printf("%d\n", (b && c) == (b || ! c &&a) );
}
```

Ejercicio 14. Operador condicional

Determinar la salida a pantalla del siguiente programa C

```
#include <stdio.h>
void main(  )
 {
   int i=2,j;
   printf("%d\n",i>=2?3:4);
   j=i+4<5?3:7;
   printf("j= %d\n",j);
}
```

Ejercicio 15. Mezclamos todos los operadores

Para afianzar todos los conocimiento referentes a los distintos tipos de operadores, prioridades, conversiones, etc, se pide determinar la salida a pantalla del siguiente programa C (se recomienda hacerlo con todo detalle), sabiendo que ócô tiene código 99 y ódô código 100:

```
#include <stdio.h>
void main( )
{
   int i=8,j=5;
   float x=0.005,y=-0.01;
   char c='c',d='d';
   printf("%d\n ",(3*i-2*j)%(2*d-c));
   printf("%d\n",2*((i/5)+(4*(j-3))%(i+j-2)));
   printf("%f\n",(i-3*j)%(c+2*d)/(x-y));
   printf("%d\n",-(i+j));
   printf("%d\n",++i);
   printf("%d\n",i++);
   printf("%d\n",--j);
   printf("%f\n",++x);
   printf("%f\n",y--);
   printf("%d\n",i<=j);
   printf("%d\n",c>d);
   printf("%d\n",x>=0);
   printf("%d\n",x<y);
   printf("%d\n",j!=6);
   printf("%d\n",c==99);
   printf("%d\n",5*(i+j)>'c');
   printf("%d\n",(2*x+y)==0);
   printf("%f\n",2*x+(y==0));
   printf("%d\n",2*x+y==0);
   printf("%d\n",!(i<=j));
   printf("%d\n",!(c==99));
   printf("%d\n",!(x>0));
   printf("%d\n",(i>0)&&(j<5));
   printf("%d\n",(i>0)||(j<5));
   printf("%d\n",(x>y)&&(i>0)||(j<5));
   printf("%d\n",(x>y)&&(i>0)&&(j<5));
}
```

LECTURA DE DATOS

Ejercicio 16. Lectura de cadenas de caracteres

Diseñar un programa C que pida al usuario que introduzca su nombre y apellidos y produzca como salida:

Gracias, xxxxxxxxxxxxxxxxx. Bienvenido a la programación en C
(Donde xxxxxxxxxxxxxxxxxx es el nombre y apellidos introducidos)

Ejercicio 17. Diseñar un programa C que pida al usuario su edad, almacenándola en la variable entera *edad*, el año actual, almacenándolo en la variable *fecha_act* y su nombre, almacenándolo en la variable de caracteres de longitud 25 *nombre*. Y que para un ejemplo de datos introducidos:

27 2003 Pedro

produzca la siguiente salida de resultados:

Pedro, naciste en 1976

Ejercicio 18.

Diseñar un programa C que calcule el área de un trapecio ; los datos de entrada serán: lado superior, base y altura. El programa debe leer los datos por teclado y producir como salida (con dos cifras decimales):

El área del trapecio es xxxxxxxxxx.xx

Ejercicio 19.

Diseñar un programa C que compruebe si un valor x0 es raíz de una ecuación de segundo grado cuyos coeficientes son a, b, c; esto es, si x0 verifica la ecuación:

$$a\,x_0^2 + b\,x_0 + c = 0$$

La salida del programa a pantalla será una de estas dos líneas:
RESULTADO DE LA COMPROBACIÓN: es raiz
RESULTADO DE LA COMPROBACIÓN: no es raiz
Indicación: Se debe utilizar una expresión lógica que compruebe la condición, y mediante un operador condicional obtener la salida a pantalla adecuada.

1.6 Soluciones a las prácticas

Ejercicio 1.

Únicamente se debe realizar el seguimiento de lo explicado en el enunciado.

SALIDA A PANTALLA

Ejercicio 2.

Apartado a) Se debe incluir la secuencia de escape \n para efectuar un salto de línea.

```
#include <stdio.h>
void main ( )
{
/* Instrucción para escribir en pantalla*/
printf("Este es mi primer programa C.\nProgramar es sencillo y divertido.\n");
}
```

Apartado b) Dos sentencias printf no son equivalentes a dos líneas de escritura en pantalla, luego hemos de incluir la secuencia de escape \n de igual forma que en el apartado anterior.

```
#include <stdio.h>
void main ( )
{
/* Instrucción para escribir en pantalla*/
        printf ("Este es mi primer programa C.\n");
        printf ("Programar es sencillo y divertido.\n");
}
```

Apartado c) Para escribir texto en pantalla se puede utilizar la función puts("") de igual forma que la función printf, pero con la diferencia de que cada vez que se utilice puts se produce un salto de línea final.

```
#include <stdio.h>
void main ( )
{
```

```
/* Instrucción para escribir en pantalla*/
puts ("Este es mi primer programa C.\nProgramar es sencillo y divertido.");
}
```

Apartado d) Utilizando dos sentencias puts no se debe incluir la secuencia de escape \n .

```
#include <stdio.h>
void main ( )
{
/* Instrucción para escribir en pantalla*/
        puts ("Este es mi primer programa C.");
        puts ("Programar es sencillo y divertido.");
}
```

Ejercicio 3.

```
/* En este programa imprimimos en pantalla una cadena formada
 * por texto  y números.
 */
#include <stdio.h>
 void main( )
 {
    int n1, n2;
    n1 = 10;
    n2 = 3;
    printf("El numero %d es mayor que el numero %d.", n1, n2);
 }
```

Ejercicio 4.

La salida a pantalla es la indicada a continuación:

```
f=      3.1416
f=       3.142
f=3.1416
f=     3.14159
f=    3.141590
f=3.141590
f=3.141590
```

```
f=3.1416
n=0007
n=   0007
```

TIPOS DE DATOS

Ejercicio 5.

A continuación se indica la salida a pantalla:

```
El caracter M como letra es M
El caracter M como numero es 77
Podemos operar con la letra
La letra + 1 vale= 78
La letra + 1 como caracter vale= N
```

Ejercicio 6.

La salida a pantalla es la siguiente:

```
El codigo ASCII para la letra A es 65
El codigo ASCII 48 representa al caracter 0
```

Ejercicio 7.

La salida a pantalla es la ofrecida a continuación:

```
Perdida de decimales. i1 solo almacena enteros. i1=10
El resultado de dividir 10 entre 3 es 3
Al asignar a x1 el valor de i1, x1 queda 10.000000
Podemos sumar un entero y un real:x1+i1=20.000000
Entero asignado a un caracter: 3 a 3 (DŽ
aritmetica entera: 3 / 2 = 1.000000
aritmetica en punto flotante: 3 / 2. = 1.500000
```

Al compilar el programa nos avisa de una posible pérdida de datos en cada conversión:
warning C4244: 'initializing' : conversion from 'const double' to 'int', possible loss of data
warning C4244: '=' : conversion from 'int' to 'float', possible loss of data
warning C4244: '=' : conversion from 'int' to 'float', possible loss of data
warning C4244: '=' : conversion from 'double' to 'float', possible loss of data

OPERADORES

Ejercicio 8.

La salida a pantalla es la siguiente:

a=0.000000 b=2.000000 c=0.000000 d=10.000000
i=2 j=3 k=9 m=11

Solución paso a paso:
a=2/3*3=(2/3)*3=0*3=0, como a es real a=0.
i=2./3*3=(2./3.)*3=2., como i es entero i=2 en este caso la división no es entera, por lo que al dividir y multiplicar por el mismo número, el factor 2. no varía.
b+=a/j; quiere decir b=b+a/j=2.+0./j=2.+0.=2.
k=5.4+b*2=5.4+2*2.=5.4+2.*2.=5.4+4.=9.4 como k entero k=9
d=i+k/i*i=2+9/2*2=2+(9/2)*2=2+4*2=2+8=10 como d es real d=10.
m=i+k/d*d=i+k=2 +9.=11. como m es entera m=11
c=1/2+1/2=0+0=0 como c es real c=0.

Ejercicio 9.

Para el compilador Visual C ++, la longitud asignada a cada uno de los tipos de datos, en *bytes*, es la indicada en la salida a pantalla de este ejercicio:

Tipo	Bytes
char	1

int	4
float	4
double	8

Ejercicio 10.

A continuación se indica la salida a pantalla:

VALORES INICIALES: x = 2; y = 3
PREFIJO: El resultado de ++x + y = 6
(Despues del calculo, x = 3 y y = 3)
VALORES INICIALES: x = 2; y = 3
SUFIJO: El resultado de x++ + y = 5
(Despues del calculo, x = 3 y y = 3)

Ejercicio 11.

La salida a pantalla es la siguiente:

1 ..... 1 es distinto de 2, por lo tanto la condición lógica es verdadera (1)
0 ..... óaô toma el valor 4 que no es igual a 2 (falso)
4 ..... óaô toma el valor 4
1 ..... óxô toma el valor 2 que equivale a 2 (verdad)
0 ..... óaô vale 4 y no es menor que óxô que vale 2 (falso)
5 ..... óxô vale 5
1 ..... óxô vale 5 y es mayor o igual a óaô que vale 4 (verdad)

Ejercicio 12.

El orden de prioridad entre los operadores que aparecen en este programa es: (de arriba abajo)

!
Relacionales (<, <=, >, >=)
Igualdad (= =,!=)
&&
||

Siguiendo el orden de prioridad establecido para la realización de las operaciones, la salida a pantalla es:

```
1 0 1 0
1
0
```

Ejercicio 13.

La salida a pantalla es la siguiente:

```
1
1
0
```

Ejercicio 14.

La salida a pantalla es la indicada a continuación:

```
3       ..... El valor de la expresión condicional es 3 puesto que i es igual que 2
j= 7    ..... A la variable j se le asigna el valor de la expresión condicional:
               como i vale 2, i+4 vale 6 y por tanto, es falso que sea menor que 5, por
               lo que el valor de la asignación es 7
```

Ejercicio 15.

Se ofrece a continuación la salida a pantalla y la explicación pertinente:

```
14      ..............  (3*8-2*5)%(2*'d'-'c')=14%(2*('c'+1)-'c'=14%('c'+2)=14%101=14
                        (el cociente de dividir 14 entre 101 es 0)
18      ..............  2*((8/5)+(4*(5-3))%(8+5-2))=2*(1+8%11)=2*(1+8)=18
-466.666677 ...  (8-3*5)%(99+2*100)/(0.005+0.01)=-7/(0.015)=-466.66667
                   % y / tienen igual prioridad: de izda a derecha
-13     ..............  -(8+5)=-13
9       ..............   ++i=++8  ; primero se incrementa quedando en i el valor 9, y es
                         lo que se imprime (el valor de la expresión)
9       ..............   i++=9++; se imprime el valor,que es 9 y después se incrementa,
```

```
                    quedando i con valor 10
4       .....       --j=--5; primero se decrementa j, quedando en 4 y se imprime
1.005000 .....      ++x=++0.005; primero se incrementa y después se imprime
-0.01000 .....      y--=-0.01--;se imprime -0.01 y después se resta 1
0       .....       i<=j ; 10<=4 ; falso (0)
0       .....       c>d ; 'c'>'d' ; 99>100 ; falso (0)
1       .....       x>=0 ; 1.005>=0 ; verdad (1)
0       .....       x<y ; 1.005<-1.01 ; falso  (0)
1       .....       j!=6 ; 4!=6 ; verdad (1)
1       .....       c==99 ; verdad (1)
0       .....       5*(i+j)>'c' ;  5*(10+4)=70>'c'  ; falso (0)
0       .....       (2*x+y)==0 ; 2*1.005-1.01==0 ; falso (0)
2.010000 .....      2*x+(y==0) ; 2*1.005+falso ; 2*1.0005+0=2.001
0       .....       2*x+y==0 ; (2*1.005-1.01)==0 ; falso (0)
1       .....       !(i<=j) ; !falso=verdad (1)
0       .....       !(c==99) ; !(verdad) ; !1=0 ; falso (0)
0       .....       !(x>0) ; !(verdad) ; !(1)=0  falso (0)
1       .....       (i>0)&&(j<5) ; V&&V=verdad  (1)
1       .....       (i>0)||(j<5) ; V||V=verdad  (1)
1       .....       (x>y)&&(i>0)||(j<5) ; V&&V||V ; (V&&V)||V=verdad (1)
1       .....       (x>y)&&(i>0)&&(j<5) ; V&&V&&V=verdad (1)
```

LECTURA DE DATOS

Ejercicio 16.

Un programa válido es el siguiente:

```
#include <stdio.h>
void main(  )
{
        char nombre[50];
        printf ("Escriba su nombre y apellidos y pulse la tecla INTRO\n");
        scanf  ("%[^\n]", nombre);
        printf ("Gracias, %s. Bienvenido a la programacion en C\n", nombre);
}
```

La sentencia de lectura que emplea la función scanf podría sustituirse por:

```
gets(nombre);
```

Ejercicio 17.

A continuación se ofrece una solución correcta:

```
#include <stdio.h>
void main ( )
{
        int edad, fecha_act;
        char nombre[25];
        printf ("Introduce tu edad: ");
        scanf ("%d",&edad);
        printf ("Introduce el año actual: ");
        scanf ("%d",&fecha_act);
        printf ("Introduce tu nombre: ");
        scanf ("%s",nombre);
        printf ("%s, naciste en %d.\n", nombre, fecha_act-edad);
}
```

Ejercicio 18.

Un posible programa es el siguiente:

```
#include <stdio.h>

void main( )
{
   float area, base, altura, lado_sup;

   printf("CALCULO DEL AREA DE UN TRAPECIO\n");
   printf("Introducir los siguientes datos:\n");
   printf("Base del trapecio: ");
   scanf("%f", &base);
   printf("Altura del trapecio: ");
   scanf("%f", &altura);
   printf("Lado superior del trapecio: ");
   scanf("%f", &lado_sup);
   area = (lado_sup + base) * altura/2;
   printf("El area del trapecio es %.2f\n", area);
}
```

Ejercicio 19.

Una solución válida es la indicada a continuación:

```
#include <stdio.h>

void main( )
{
   int es_raiz;
   float a,b,c,x0;
   printf("Verificación de raiz de ecuacion de segundo grado\n");
   printf("Introducir los tres coeficientes de la ecuacion ax2+bx+c\n");
   scanf("%f %f %f", &a, &b, &c);
   printf("Introduce el valor de la posible raiz: ");
   scanf("%f", &x0);
   es_raiz=a*x0*x0+b*x0+c= = 0;
   printf("RESULTADO DE LA COMPROBACION: %s\n",
                                        es_raiz?"es     raiz":"no     es     raiz");
```