Unidad 3-4 : Introducción a la programación en C
Desarrollo de la unidad : 36 h
Prácticas : Utilizar el compilador Borland
Ejercicios : Pasar los algoritmos en pseudocódigo a Lenguaje C
Conceptos:
Historia del C, Características, Estructura del programa en C, Elaboración de un programa, declaración de datos, instrucciones, operadores, funciones del usuario, funciones de librería,

Introducción:
- Lenguajes de programación, tipos, evolución.

Origen

Creados por Dennis Ritchie y Ken Thompson a inicios de los 1972, como lenguaje para la construcción del S.O. Unix. El lenguaje C es un lenguajes estructurado que mantuvo las mejores características de sus predecesores, los lenguajes B y BCPL. Es a pesar de los años transcurridos el lenguaje más utilizado para desarrollo de sistemas: Software de sistemas operativos, comunicaciones, redes, drivers, etc.

Sin embargo no es el más indicado para el desarrollo de aplicaciones de gestión sencillas, ni el más didáctico. Posee una periodo de aprendizaje mayor que otros leguajes modernos.

Evolución: ANSI C (1983) , C++ (Bjarne Stroustrup), Visual C, Borland C, Java (Sun Microsystems), C#

Características:

- Lenguaje estructurado, que permite el diseño modular
- Lenguaje de propósito general (más enfocado al desarrollo de aplicaciones de sistemas y herramientas que aplicaciones de gestión).
- Lenguaje intermedio: Lenguaje de alto nivel con capacidades de acceso a bajo nivel
  Ej.- Tratamiento de direcciones, bits, ensamblador, acceso a funciones del sistema operativo, etc => Lenguaje muy eficiente
- Lenguaje portable o portátil: Independiente del S.0. y de la arquitectura hardware.
- Exige una programación cuidadosa, ofrece gran libertad al programador lo que se traduce en poco control semántico. Ej.- Lenguaje poco tipado y no controla los límites en las tablas.
  - No es un lenguaje para inexperto, poco didáctico.
- Posee un conjunto de normas simples y muy reducidas.
- Permite hacer programas rápidos y eficientes.
- No incluye dentro del lenguaje las operaciones de entrada y salida, tratamiento de ficheros o bases de datos, gestión de memoria dinámica: es necesario el uso de funciones de librería. (Que no forman parte del lenguaje en sí, pero siempre suelen estar incluidas)
- Permite un fácil enlace con otros lenguajes en especial como el ensamblador.

Elaboración de un programa

1.- Edición de un programa (Cualquier editor que genere fichero de texto puro)
ficheros fuentes programa.c

2.- Compilarlo (Traducción de Lenguaje C a código máquina)
programa.obj programa.asm

3.- Enlazado, Linkado, Montado
(Unir los distintos módulos y librerías para crear un fichero ejecutable.
libm.lib libreria.dll *.obj -> *.exe *.com

4.- Ejecución y prueba.

Esquema

Edición Compilación Enlace
modulo1.c ----> modulo1.obj
modulo2.c ----> modulo2.obj ---> programa.[exe | com]
moduloX.asm
librerias. lib
<recursos windows>

COMPILADORES EN C

Existen mucho programas que permiten compilar programa en C. En entorno Windows los más conocidos son los compiladores propietarios de Borlard y de Microsorf. Existen tambien compiladores de código abierto, que se pueden usar libremente sin licencia. Ej.- gcc del GNU.

También existen algunos interpretes de C que se suelen utilizar para depuración.

Modo de trabajos

- **Modo comando**: Incluyendo ordenes de compilación en el interprete de comandos.

Ej.-
Editamos un fichero con algún programa que genere ficheros de texto:
Ej,.- edit, Notepad, Wordpad, etc. Y desde una ventana de Interprete de comandos (MS-DOS) introducimos ordenes de compilación:

C\:Mis Documentos>bcc –c fichero.c

- **Entornos integrados**
En Windows: Borland y Microsoft, En GNU/Linux Kdevelopper del proyecto KDE

Formados por un conjunto de utilidades que suelen incluir:
- Editor sensible al contexto
- Compilador y Enlazador integrados
- Herramientas de depuración: Breakpoint, paso a paso, consultar el valor de variables
- Herramienta de acceso a BD
- Administración de proyectos

- Herramientas de diseño de interfaces gráficos (RAD) (Ventanas, Botones, Menús, Iconos)
- Ayuda y ejemplos

Estructura general de un programa en C

1. **Cabecera**

- Comentario general sobre el fichero. (Opcional)
  Se suele incluir el Nombre del módulo, la Descripción, Autor, Fecha, Versión, etc
  Cada empresa o proyecto puede determinar la información definir en este apartado.

  Existen herramientas de gestión de proyectos y control de versiones que determinan el formato de la cabecera.

```
/* ------------------------------------------------------------------------
      MODULO:  ControlAlarmas.c
      DESCRIPCIÓN: Realiza el análisis del las alarmas generadas por
                   el secuenciador de la central A-49/B
      AUTOR:  Jerónimo Martín.
      FECHA :  10/12/2002
      Versión:    2.0.1
------------------------------------------------------------------------*/
```

- Inclusión de librerías:
  Debemos incluir todas las definiciones de funciones que no estén incluidas dentro del el propio archivo. Normalmente primero se indican las del sistema, después la propias del proyecto

  #include <stdio.h> Definiciones de funciones de la librería de E/S estándar
  #include "secuenciador\alarmas.h" Definiciones utilidades del módulo para gestionar alarmas.
  .

- Definición de constantes y expresiones simbólicas
  # define MAXNOTA 100
  # define MENSAJE "No hay nadie "

2. **Funciones del usuarios (Subprogramas)**

Codificación de todos los subprogramas que componen el programa

/* Descripción de la funciones y de sus parámetros */
<Valor devuelto> nombre (Parámetros)

{ /* Inicio */
Definición del entorno tipos de datos y variables

Instrucciones; /* Separadas por punto y coma */

} /* Fin Función */

3. Función main o Programa Principal

```
main ( )
{
Definición de datos;
Instrucciones;
}
```

En algunos autores aparece primero la declaración o cabecera de las funciones auxiliares, después la codificación de la función main, y por último la codificación de las funciones.

Ejemplos:

```
/* Este es mi primer programa en C */
#include <stdio.h>

main()
{
  puts("Hola este es mi primer programa");
}
```

```
/* Este es mi segundo programa en C */
#include <stdio.h>

int Sumar ( int A, int B)
{
int resultado;
resultado = A + B;
return resultado;
}

main()
{
int uno, dos, valor;
uno = 1;
dos =  2;
valor = Sumar( uno, dos);
printf(" El resultado es : %d  \n", valor);
}
```

ENTORNO DE UN PROGRAMA EN C:
Definición de la estructura de datos.

Constantes

- Del prepocesador (Datos o expresiones)

Ejemplo:

```
#define PI   3.1416
#define  MENSAGE  " Como está ustedes "
#define  MAXLONGITUD   200
#define FIN        }
#define INICIO  {
#define  Mostrar( Info )   puts( Info )
```

No ocupan posiciones del memoria son sustituidas por el valor correspondiente antes de realizar la compilación.
if (Dato > MAXLONGITUD) ---→ if (Dato > 200)

- Variables constantes con tipo determinado (mediante el modifiador const)

Ejemplo:

```
const   int  nveces = 0;
```
Defino una constante entera (int) llamada nveces cuyo valor es 0

Ocupan posiciones de memoria y se pueden consultar con en depuración

TIPOS DE DATOS EN C

ESTRUCTURAS INTERNAS:
Aquellas que se almacenan en memoria principal

Estructuras estáticas
No cambian de estructura, tamaño o número de elementos

Tipos Simples (Un único elemento)

Ordinales:
enteros : **int** (signed, unsigned, short, long)
caracteres: **char** (signed, unsigned)
enumerados: **enum**

El tipo Lógico no existe
(0 es falso cualquier otra cosa se considera verdadero)

Reales : **float**, double

Punteros: Variables que guardan direcciones de otras variables.
<Tipo base> * nombre; Ej- int *pdato;

Tipo void: Tipo vacio o nulo

Tipos Compuestos (varios elementos)

Tablas
Tablas unidimensionales
<Tipo base> nombre [<Tamaño>] Ej.- int Datos [10];
Tablas bidimensionales
<Tipo base> nombre [<Tamaño>] [< TamañoC>] Ej.- int Matriz [4][10]

String o Cadenas : Tablas de caracteres con el terminador '\0'
Ej.- char Nombre [100] = "Hola Mundo";

Registros o estructuras (Varios elementos de distinto tipo)
struct

Uniones
(Varios elementos de distinto tipo sobrepuestos en el mismo espacio de memoria)
union

Estructuras dinámicas:

No existen estructura dinámicas predefinidas, las construyen el programador a su medida utilizando punteros y memoria dinámica. Los tipos más comunes son: listas, colas, pilas, árboles y grafos.
Ej.-

```
int *pdato;
pdato = (int *) malloc( 10* sizeof(int));
/* pdato señala a una zona de memoria reservada para 10 enteros */
```

Estructuras externas

El manejo de estructuras externas no está predefinido en el lenguaje C, aunque existe una librería estándar para realizar operaciones básicas sobre Ficheros :<stdio.h>
Existen funciones para manejar dos tipos de ficheros:

- Ficheros de texto.
    - Ficheros que contienen secuencias de caracteres separados por saltos de línea
    - Se realiza operaciones de lectura y escritura carácter a carácter o línea a línea.

- Ficheros planos o Binarios
    - Cualquier tipo de fichero sin especificar su estructura.
    - Se realiza operaciones de lectura, escritura y posicionamiento a nivel de byte.

ELEMENTOS SINTÁCTICOS DEL LENGUAJE C (TOKENS)

Existen seis clases de *componentes sintácticos* o *tokens* en el vocabulario del lenguaje C: ***separadores, identificadores, palabras claves, constantes y operadores.***

Separadores
Sirven para diferenciar / separar los elementos
Espacios, salto de línea, puntuación (, ;), tabuladores

Identificadores : (Nombres de variables, funciones y estructuras de datos definidos por el programador.

Reglas

1. Un ***identificador*** se forma con una secuencia de ***letras*** (minúsculas de la ***a*** a la z; mayúsculas de la ***A*** a la Z; y ***dígitos*** del ***0*** al 9).
2. El carácter ***subrayado*** o ***underscore*** (_) se considera como una letra más.
3. Un identificador no puede contener espacios en blanco, ni otros caracteres distintos de los citados, como por ejemplo `(*,;.:-+,` etc.).
4. El primer carácter de un identificador debe ser siempre una letra o un (_), es decir, no puede ser un dígito.
5. Se hace distinción entre letras mayúsculas y minúsculas. Así, **Masa** es considerado como un identificador distinto de **masa** y de **MAS**A.
6. ANSI C permite definir identificadores de hasta 31 caracteres de longitud.

Ejemplos de identificadores válidos son los siguientes:

```
tiempo, distancia1, caso_A, PI, velocidad_de_la_luz
```

Por el contrario, los siguientes nombres no son válidos (¿Por qué?)

```
1_valor, tiempo-total, dolares$, %final
```

Palabras clave o reservadas
Palabras predefinidas en C con un significado propio, que no podemos utilizar como identificadores.

```
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
```

Constantes
(Representan valores concretos dentro de una definición o expresión)

- Numéricas (Entero o real) 124 -34.45 8.04e-3
- Caracteres ‘A’. ‘*’, ‘ ‘, ‘\n’ ‘\t’ →Entre comillas simples
- Cadenas de caracteres “Hola como estás” → Entre comillas dobles

Operadores
(Sirven para construir expresiones y sentencias)

Muy importante:
No confundir el operador de asignación (= , un igual) con el operador de comparación (= =, dos iguales).

- Asignación: =
- Aritméticos: +, -, *, / , % (mod),
  - Incremento y Decremento: ++, --,
    - Ej.-
    - Num++ → Num = Num + 1
  - Acumulación: +=, -=, *=, /= (*Desaconsejados*)
    - Num += 3 → Num = Num + 3
- Relacionales : >, <=, >=, = =, <>, < ,>
- Lógicos: &&(and), || (or), ¡(not)
- Binarios: >>,<<, |, &, ^, ~
- Operadores direcciones de memoria: & (Dirección de) y * (Lo señalado por)
- Operador condicional: (condición)? A: B
  Ej.- num = (A > 0)?10:-10;

Comentarios:
/* Soy un comentario en formato clásico */
// Hola soy un comentario ansi Co C++

TIPOS DE DATOS SIMPLES

ENTEROS

Tipos básicos

int
unsigned int
signed int
long int
short int

Siempre se cumple:
short < int < long

Tamaño según sistema :
- Corto 16
Con 16 bits se pueden almacenar 2^{16} = 65536 números enteros diferentes: de 0 al 65535 para variables sin signo, y de -32768 al 32767
- Largo 32
Con 4 bytes (32 bits) para almacenarlos, por lo que se pueden representar 2^{32} = 4.294.967.296 números enteros diferentes.
Si se utilizan números con signo, podrán representarse números entre -2.147.483.648 y 2.147.483.647.
BORLAND : long y int = 4 bytes y short de 2 bytes

REALES
Representación en coma flotantes

Reales en arquitectura PC

float 4 bytes 32 bits 24 mantisa 6 dígitos de precisión, 8 exponente +/- 127
double 8 bytes 64 bits 63 mantisa 12 dígitos de precisión, 11 exponente +/- 1024

Valores de números reales

```
1.23      constante tipo double (opción por defecto)
23.963f   constante tipo float
.00874    constante tipo double
23e2      constante tipo double (igual a 2300.0)
.874e-2   constante tipo double en notación científica (=.00874)
.874e-2f  constante tipo float en notación científica
```

seguidos de otros que no son correctos:

```
   1,23 error: la coma no está permitida
  23963f error: no hay punto decimal ni carácter e ó E
     .e4 error: no hay ni parte entera ni fraccionaria
```

Cuando se mezclan tipos (enteros y reales) el resultado se convierten al valor más grande.

Lectura y Escritura básica de números entero y reales:

- Leer y mostrar un número entero:

```
int numI;

scanf ("%d",& numI); // Formato y dirección del la variable a rellenar
printf("%d", num1);    // Formato y valor a mostrar
```

- Leer y mostrar un número entero:

```
float numR;

scanf ("%f",& numR); // Formato y dirección del la variable a rellenar
printf("%f", numR);    // Formato y valor a mostrar
```

CARACTERES

char letra; Ocupan un byte, Almacenan una carácter según el código ASCII (7 y Extendido)

letra = ‘a’ Un carácter entre comillas simples
letra = ‘ ‘ Carácter espacio

Orden ‘A’ < ‘ Z’ < ‘a’ < ‘z’

Internamente son números y se pueden operar como tal

letra = ‘a ‘; letra++; letra == ‘b’
letra = ‘B’; letra = letra + (‘a’ – ‘A’); letra == ‘b’ A minúsculas toupper()
letra = ‘b’; letra = letra - (‘a’ – ‘A’); letra = ‘B’ A mayúsculas tolower()

Constantes:
‘a’, ‘\0xA2’, ‘\023’: Letra, código hexadecimal, octal

Caracteres especiales

sonido de alerta	\a 7	salto de página	\f 12
nueva línea	\n 10	barra invertida	\\ 92
tabulador horizontal	\t 9	apóstrofo	\' 39
tabulador vertical	\v -	comillas	\" 34
retroceso	\b 8	carácter nulo	\0 0
retorno de carro	\r 13		

Ejemplo:

```
/* Uso de las secuencias de escape */
#include <stdio.h>
main() /* Escribe diversas sec. de escape */
{
  printf("Me llamo \"Nemo\" el grande");
  printf("\nDirección: C\\ Mayor 25");
  printf("\nHa salido la letra \'L\' ");
  printf("\nRetroceso\b");
  printf("\176");                    /* corresponde en binario a 126 (~) */
  printf("\n\tEsto ha sido todo");
}
```

Leer y escribir caracteres

- Funciones de la librería: **stdio.h**
  Entrada con buffer de la entrada estándar
  scanf(“%c”, & letra);
  printf(“ %c “, letra);
  letra = getchar();
  putchar(letra);

- Funciones de la librería: **conio.h**

Entrada directa de la terminal / consola
getch(); Sin eco
getche(); Con eco
putch(letra);

Ejecutar los programas desde una pantalla de MS-DOS

Ejercicios a probar:
Contar caracteres, contar espacios, contar vocales, contar mayúsculas y minúsculas , mostrar sólo las letras mayúsculas

ENUMERADOS

typedef enum { lunes, martes, miercoles, jueves, sabado, domingo } diasemana;
diasemana dia1,dia2;

Ojo: no se permiten acentos ni caracteres especiales.

typedef enum { rojo, verde, amarillo } semaforo; -> enum semaforo { rojo, verde, amarillo } semaforo f1, f2;

No se pueden leer ni escribir directamente, para uso interno, su representación en memoria es simplemente un número entero el primer valor se le asigna por omisión el 0 y el resto 1,2,...N

typedef enum { GRANDE=1, MEDIANO, PEQUENIO } tallas;
typedef enum { rojo=3, verde=1, amarillo=4, azul=9, negro=10, blanco=0 } colores;
Se pueden operar y comparar como números enteros

RESUMEN DE TIPOS DATOS SIMPLES

TIPO	**Esp.**	**LONGITUD**	**RANGO**
unsigned char	%uc	8 bits	0 a 255
char	%c	8 bits	-128 a 127
unsigned int	%u	16 bits (32 bits)	0 a 65.535 (4.294.967.295)
short int	%hd	16 bits	-32.768 a 32.767
int	%d	16 bits (32 bits)	-32.768 a 32.767 (2.147.483.648)
unsigned long	%lu	32 bits	0 a 4.294.967.295
long	%ld	32 bits	-2.147.483.648 a 2.147.483.648
float	%f	32 bits	3,4 x e-38 a 3,4 x e+38
double	%lf	64 bits	1,7 x e-308 a 1,7 x 1e+308
long double	%Lf	80 bits	3,4 x e-4932 a 3,4 x e+4932

TIPO VOID (vacío)

Sin valor, tipo nulo vale para definir los parámetros y resultados de las funciones

MODIFICADOR const

const int dato = 10;
Podemos indicar que una variable tenga valor fijo y que no se pueda modificar con en prefijo const, (útil en la para la definición de parámetros de la funciones) a diferencia de #define ocupa memoria

INSTRUCCIONES Y SENTENCIAS DE CONTROL

Instrucción secuencial Simple / Compuesta

- Simple : Instrucción; (Siempre termina con punto y coma)
- Compuesta o Bloque de instrucciones (Entre llaves)

```
{
   Instrucción;
   Instrucción;
}
```

En todas las sentencias alternativas o repetitivas es siempre preferible utilizar un bloque de intrucciones aunque sólo se incluya una instrucción.
El formato:

```
if ( Contador > 0 )
  {
  Suma = Suma + Contador;
  }
```

Es Preferiblea a:

```
if ( Contador > 0 )
    Suma = Suma + Contador;
```

Sentencia alternativa simple SI

if (Condición**)**
  Instrucción S/C
[**else**
  Instrucción S/C]

```
if (expresion_1)
sentencia_1;
else if (expresion_2)
sentencia_2;
else if (expresion_3)
sentencia_3;
else if (...)
...
[else
sentencia_n;]
```

Cuando hay varios if anidados se supone que el else es del if más próximo.

Ejemplos:

```
/* Simula una clave numérica de acceso, con un único intento */

#include <stdio.h>
main(){
  unsigned long Usuario, Clave=18276;
  printf("Introduce tu clave: ");
  scanf("%d",&Usuario);
  if ( Usuario == Clave )
        {
        printf("Acceso permitido");
        printf("Enhorabuena");
        }
 else
        {
        printf("Acceso denegado");
        }
}
```

```
/* Escribe bebé,,niño, joven o adulto */
#include <stdio.h>
main()
{
  int edad;
  printf("¿Cuántos años tienes?: ");
  scanf("%d",&edad);
  if (edad<1)
        printf("No es una edad correcta.");
  else if (edad<3)
        printf("Eres un bebé,");
  else if (edad<13)
        printf("Eres un niño");
  else if (edad<18)
        printf("Eres un joven");
  else
        printf("Eres un adulto");
}
```

Sentencia alternativa múltiple (*SEGÚN)*

```
switch (expresión) {
case expresión_cte_1: sentencia_1;
                      sentencia_12;
                      [break]
case expresión_cte_2: sentencia_2;
      ...
case expresión_cte_n: sentencia_n;

[default: sentencia;]
}
```

El funcionamiento de la instrucción switch es un poco peculiar:
Si la expresión coincide con algún caso, se ejecutan las instrucciones de la parte derecha y se continua con las instrucciones de los siguientes casos a no ser que exista una instrucción **break** que provoque la salida del switch.

Ejemplo

```
#include <stdio.h>
main() /* Escribe el día de la semana */
{
  int dia;
  printf("Introduce el día de la semana (1 al 7): ");
  scanf( "%d" , & dia );
  switch (dia )
    {
     case 1: printf("Lunes"); break;
     case 2: printf("Martes"); break;
     case 3: printf("Miércoles"); break;
     case 4: printf("Jueves"); break;
     case 5: printf("Viernes"); break;
     case 6: printf("Sábado"); break;
     case 7: printf("Domingo"); break;
     default: printf(" - Valor erróneo -");
    }
}
```

Ejercicio: Dada las siguientes instrucciones, indicar cual seria las funciones ejecutadas según los valores de la variable letra = A,a,B,b,C,c,t,M,R

```
switch ( letra )
{
  case 'A':
  case 'a':  menu A ();
             break;
  case 'B':  menu B()
             break;
  case 'b':  menub()
             break;
case  'C': menuC();
case 'c'  : menuc();
             break;
case 't'   : menut();
             break;
default    : menuX();
}
```

Solución:
'A' menuA()
'a'-> menuA()
'B'-> menuB()
'b'->menub()
'C'-> menuC() y menuc()
'c'-> menuc()
't'-> menut()
'M'-> menuX()
'R'-> menuX()

SENTENCIAS REPETIVAS

MIENTRAS

```
while ( Condición )
   {
    Instrucción S/C
   }
```

REPETIR (OJO: mientras no *hasta*)

```
do
  {
  Instrucciones;
  }
while ( Condición );
```

Ejemplos :

Num= 5;
while (num > 0)
  {
  putchar('*');
  num--;
  }
	num = 5;
	while (num--)
	  {
	  putchar('*');
	  }
	
	/* Cuando num vale cero
	termina , 0 es falso */
		num = 5;
		do
		  {
		  putchar('*');
		  num--;
		  }
		while (num > 0)
		/* Si num = 0 se hace */

```
// Ciclo while infinito
while ( 1 )
      {
      HacerAlgo();
      }
```

CICLO PARA

```
for  ( Valores iniciales;  Condición ; Incrementos )
            Instruccion;
```

Ejemplos:

for (i= 1; i <= 10; i++)
  {
  printf(" %d ",i);
  }
 	
Resultado:
1,2,3,4,5,6,7,8,9,10
	for (i = 10; i > 0; i--)
	  {
	  printf("%d",i);
	  }
	
	Resultado:
	10,9,.8,7,6,5,4,3,2,1
		for (i=0,j=5; i < j; i++, j--)
		  {
		  printf(" %d : %d \n ", i,j);
		  }
		
		Resultado
		0:5 , 1:4 , 2:3

Formatos raros:

for (; i > 0 ;)
{
i--;
}
/* No hay inicialización, ni el incremento */
	for (; ;)
	{
	HacerAlgo();
	}
	/* Ciclo infinito */

- **Sentencia continue:**

Esta instrucción, cuando aparece dentro de cualquier ciclo (while, do while, for), salta la sentencias que se repiten y vuelve a evaluar la condición del ciclo.

```
#include <stdio.h>
/* Escribe del 1 al 100 menos los múltiplos de 11 */
void main()
{
  int Num =1;
  while  (Num <=100)
    {
    if   ( (Num % 11)==0)
          {
          Num ++;
         continue;
          }
    printf("%d\n", NUM);
    Num ++;
    }
}
```

- **Sentencia break**

Esta instrucción provoca la salida la terminación inmediata del ciclo.

```
#include <stdio.h>
/* Calcula la  suma100 número, pero si encuentra el número cero termina inmediatamente */
void main()
{
  int Num, Suma, i ;
  Suma = 0;
  for (i=1; i < =100; i++ )
     {
     scanf("%d", & Num );
     if  ( Num == 0 )
         {
         break;
         }
     Suma = Suma + Num;
     }
  printf(" El resultado es = %d \n", Suma);
}
```

OPERADORES ENTEROS A NIVEL DE BIT

El lenguaje C es un acceso a bajo nivel lo que permite trabajar a nivel de bit.
Operadores de bit:

AND (&)	Enmascarar	valor = valor & 0x00FF (Se queda con el byte bajo)
OR (|)	Activar un bit	valor = valor | 0x00004 (Activa el 3º bit [4-> 100])
XOR (^)	Bit diferentes	distintos = 0x00A3 ^ dato
NOT (~)	Invertir los bit	dato = ~dato;
Desplazamiento Dcha (>>)		Mover un bit para preguntar por él
Desplazamiento Izda (<<)		

Desplazar a la Dcha (>>) es lo mismo que dividir entre dos.
Desplazar a la Izda (<<) es lo mismo que multiplicar por dos.

Ejemplos:

1.- Obtener el valor de cada byte en de un entero corto.

```
Short int valor;
ByteAlto =  valor >> 8;
ByteBajo =  valor  & 0x00FF;  /* Otra forma   ((valor  << 8) >> 8) */
```

2.- Poner el 3º y 5º bit a uno: (14_{16}) = (10100_{2})

```
valor =  valor |  0x0014
```

3.- Contar el número de bits distintos entre la variables de tamaño byte

```
mascara =  dato1 ^ dato2;
nbits = 0;
for (i= 0 ; i < 8 ; i++ )
    {
    if  ( mascara  &  ( 0x01 << I ) )
        {
         nbits++;
         }
    }
```

4.- Mostrar el valor binario de un número

```
 for(I=15; I>=0 ;I--)
  {
    dato=0x0001<<I;
    if(entero & dato)
      {
       putchar('1');
      }
    else
      {
      putchar('0');
      }
  }
```

5.- ¿Cuál es el valor final de la variable dato? Respuesta: El valor 7

a)
```
dato = 2;
dato  +=  0x0005 |   ( 0x0001 << 2 );
```

TIPO PUNTERO

Punteros y direcciones de variables

El manejo de punteros en el lenguaje C es una capacidad que ofrece una gran potencia y flexibilidad, a la hora de realizar programas complejos. Sin embargo su manejo es delicado y es una fuente habitual de errores, que en muchos cosas pueden ser muy graves. En la mayoría de los lenguajes de alto nivel no existe esta capacidad o se encuentra muy limitada. En lenguajes modernos como Java, no existen los punteros como tales, sustituyéndose por una sintaxis especifica en funciones y memoria dinámica.

Los punteros surgen de la necesidad de conocer la dirección de memoria de una variables, más que el valor que tiene.

Definición de un puntero
TipoBase * NombreVarPuntero

Ej .- int *pnum;
char * señalaletra;
float *lugarcifra;

Operaciones
int dato1, dato2;

dato1= 0;
dato2 = 10;
/* Asignar un valor */
pnum = & dato1; /* pnum almaceno la dirección de dato */

*pnum = 23 **;** /* Donde señala pnum se almacene un 23 , Es equivalente a dato1 = 23 */

Hay que tener muy presente cuando accedemos al valor del puntero (una dirección) y al valor que esta señalando (el valor del la variable señalada):
char *pc, letra;
pc = & letra; /* Debemos poner una dirección de una variable tipo carácter */

***pc = 'a'**; / * Debemos poner una expresión de tipo carácter */

El valor NULL, Para indicar que un puntero no señala a ningún situo existe la constante NULL
if (puntero == NULL)
puts("No hay dato");

Nota:¿Por qué scanf necesita el operador & y printf no?. La función scanf necesita la dirección de la variable donde va a guardar el nuevo valor, (para scanf es un parámetro de salida , que debe ser pasado por referencia). La función printf sólo accede al valor (utiliza el parámetro sólo de entrada)

int dato = 100;

scanf("%d", & dato);
printf("%d", dato) ;

Si pusiese printf("%d", & dato) → Se mostraría la dirección de memoria donde se almacena la variable dato.
Si pusiese scanf("%d", dato)→ Guardaría el valor de la leído el la dirección que tiene dato en este caso el la posición de memoria 100, lo que generalmente provocaría un error de ejecución.

SUBPROGRAMAS EN C

Formato:
<tipo de resultado devuelto> nombre de la función (parámetros)
parámetros = tipo nombre, tipo nombre, Formatos antiguos
{

}

*No existen en C los procedimiento sino funciones que no devuelven nada (**void**)*

Lugar de definición de las funciones:
La funciones se suelen definir antes o después de main(). Para evitar problemas de compilación lo mejor es colocar las todas las cabeceras de las funciones antes de la implementación de cualquier función.

Ejemplos de funciones:

```
float Potencia ( float base, int potencia )
void  PintaCuadrado ( int tamano );
int   ElMayor ( int dato1, int dato2, int dato3)
void InicializaT ( int tabla[10], int valor )
void TablaMultiplicar ( int Num );
```

Tipo de parámetros :
Según el uso: **Entrada, Salida, Entrada y Salida**
Según la información que se transfiere: **Por copia o Referencia** (dirección o puntero al dato)

Ejemplo:

```
void  incrementar1 ( int dato, int valor )
{  dato = dato + valor; }

void  incrementar2 ( int *pdato , int valor)
{ *pdato = *pdato + valor );

void main()
{
int num= 0;
incrementar1 (num,4); // La variable num, no se altera, paso por copia
incrementar2 (&num,4); // La variable num, sí se altera, paso por referencia
}
```

¿ Cuando utilizar el paso por referencia ?
- ➔ En los parámetros de salida o E/S, siempre se debe pasar la dirección del dato no su valor o copia
- ➔ Cuando el tamaño del parámetro es muy grande Ej- Una tabla siempre se pasa por referencia.

LAS FUNCIONES : printf y scanf

Mientras que otros lenguajes de programación utilizan instrucciones especificas para realizar las operaciones de entrada y salida, C lo hace utilizando funciones de librerías. Por lo que para poder utilizarlas tendremos que incluir en nuestro programa el fichero cabecera en el que están definidas: **stdio.h**.

Por ello todo fichero que use estas funciones deberá acceder a dicha librería incluyendo la línea:

#include <stdio.h>

En el caso de E/S por teclado y pantalla, las dos funciones de E/S con formato, más importantes son:

***printf*:** que visualiza en pantalla datos cuyo formato de salida queda definido en la propia función. Su prototipo es:

int printf (char *Formato, lista_de_argumentos....)

El valor entero corresponde al nº de caracteres escritos y es negativo (EOF) en caso de error.

Formato es una cadena de caracteres entre comillas dobles, que está formada por una combinación de caracteres y especificadores de formato, que se corresponden con cada uno de los argumentos. Si existen más argumentos que especificadores, los argumentos en exceso se ignoran. Si existen más especificados que argumentos o estos no coinciden el el tipo, la función puede provocar un grave error de ejecución .Estos especificadores de formato se construyen de la siguiente forma:

%[flags][AnchoCampo].[Precision]Type[SecuenciaEscape]

Flags Para justificar la salida. Si el AnchoCampo reservado es mayor que el dato, éste se justifica a la derecha, a no ser que usemos un – en cuyo caso se justifica a la izquierda, quedando en blanco el resto de caracteres.

AnchoCampo Nº total de caracteres que queremos reservar.

Precision Para nº decimales que queremos visualizar.

Type especifica el tipo de dato que se va a visualizar:

%d ó %i para los enteros.
%f para los float (reales en coma flotante, formato decimal)
%e float en notación científica (reales en forma exponencial)
%g usa %e o %f, la que sea más corta de las representaciones
%ld para los long int
%lf para los double
%Lf para los long double
%c para los char (un único caracter)
%s para las cadenas
%u para los enteros sin signo
%p para los punteros
%o para los datos representados en octal
%x para los datos representados en hexadecimal
%% imprime un signo %
Otros: %le, %lg, para los double en los e y g; %hu para short unsigned; %lu long unsigned...
Type es obligatorio que aparezca al igual que el %

La lista de argumentos está formada por las variables y constantes que se quieren visualizar. Deberá haber el mismo número de especificadores de formato como argumentos tenga la lista. En el caso de las cadenas, este formato, por ejemplo %5.7s visualiza la cadena con al menos 5 caracteres de largo y no más de siete. Si la cadena es más larga, el ordenador sólo mostrará los 7 primeros caracteres.

Ejemplos:

```
printf("<%5d>", 201)              < __201>  Ajuste a la Dcha
printf("<%-5d>", 201)             < 201__> Ajuste a la Izda
printf("<%2d>",201)               <201>    No hace caso de tamaño
printf("<%4.2d>",201)             <_201>   No hace caso del punto decimal
printf("<%-8.2f>",123.234)        <123.23__>  Ajuste a la Izda, mostrando 2 decimales
printf("<%5.2f>",3.234)           <_3.23>    Ajuste a la Dcha mostrando 2 decimales
printf("<%10s>","hola")           <______hola> Ajuste al a Dcha
printf("<%-10s>","hola")          <hola______> Ajuste a la Izquieda
printf("<%5.7s>","hola pepe")     <hola pe> Muestra como máximo 7 caracteres
printf  (">%d \n >%c \n",5,  'c'); >5
                                  >c

int a=20,b=10;
printf("<Por tanto %d+%d=%d>",a,b,a+b);   <Por tanto 20+10=30>
```

ii) *scanf*: que se utiliza para recoger datos del teclado y almacenarlos en una variable. Su prototipo es el siguiente:

int scanf (char *CadenaDeControl, lista_de_argumentos)

CadenaDeControl está formado por especificadores de formato, construidos como con printf.

lista_de_argumentos está formada por las **direcciones** de las variables en las que se van a almacenar los datos que se recojan desde el teclado

Ejemplo:

```
int x;
printf ("Introduce un entero:");
        scanf ("%d",&x);
```

Ejemplo:

```
/* Uso de la sentencia scanf. */
#include <stdio.h>
main() /* Solicita dos datos */
{
        char nombre[10];
        int edad;
        printf("Introduce tu nombre: ");
        scanf("%s",nombre);
        printf("Introduce tu edad: ");
        scanf("%d",&edad);
}
```

scanf y el control de errores

scanf devuelve el número de valores obtenidos, en caso de fallo devuelve 0, debemos vaciar el buffer de entrada antes de volver a llamar al scanf.

```
 int num;

do
  {
  puts("Introduce un número entero:");
  fflush(stdin);
  }
while ( scanf("%d", &num ) != 1 );
```

Solución de problemas con el buffer intermedio de teclado

Cuando leemos de la entrada estandar (stdin) a partir de la funciones de la librería stdio.h, el programa no realiza una lectura directa del teclado sino a través del buffer. El sistema operativo se encarga de leer las teclas y rellenar el buffer, este no se encuentra completo hasta que el usuario pulsa INTRO, con lo que se transfiere el control al programa.

Solución.

- Vaciar el buffer de entrada antes de realizar cualquier lectura. fflush(stdin)
- Leer hasta el salto de línea, mediante llamadas getchar();
- Utilizar las funciones de la librería conio.h que realizan la entrada directa desde teclado, aunque con esto provocamos que el programa no sea portable a otros sistemas.

ÁMBITO Y DURACIÓN DE LAS VARIABLES:

En Lenguaje C toda variable tiene que estar declarada antes de usarla.
Ej.-
int dato;

dato = 3; // Previamente hemos declarado la variable dato como de tipo int

El ámbito de una variable es el espacio dentro del programa desde donde la variable es accesible, desde este punto de vista las variables se clasifican en:

Variables locales:
Aquellas que están declaradas dentro de una función y por lo tanto sólo son accesibles dentro de dicha función.

Variables globales:
Aquellas que están declaradas fuera del cuerpo del cualquier función, por lo que son accesibles en todos los sitios del programa.[1]

Dos variables se pueden llaman igual si una variable es global y la otra local, prevalece la variable local.

La duración de una variable es el tiempo que está ocupando espacio en el la memoria del ordenador. Según este criterio podemos clasificar las variables en:

Variables automáticas o temporales:
Son aquellas variables, siempre locales, que se crean automáticamente cuando se llama a la función que las contiene y desaparecen cuando la función termina.

Variables estáticas o permanentes:
Son aquellas variables, generalmente globales, que se crean cuando el programa se carga en la memoria y permanecen durante toda la ejecución del programa.

Ejemplo 1. Casos generales

```
#include <stdio.h>
int dato;
void Saludar ( int veces)
{
 int cont;
 dato = 0;
 for (cont =0 ; cont < veces; cont ++)
     {
     puts("Hola");
      }
}
```

```
void main()
{
 int valor;

  valor = 10;
  dato = valor;
  Saludar(3);
   printf(">%d", dato);
}
```

El programa mostraría:
Hola
Hola
Hola
>0

Variables globales: dato.
Variables locales : veces y cont de la función Saludar y valor de la función main.
Variable estáticas: dato

[1] Si el programa está compuesto de varios archivos puede que la variable sea global en todos el programa o exclusivamente en el módulo o archivo donde se encuentra declarada, si añadimos el modificador static.

Variables automáticas: veces, cont y valor.(Esta variable tendría sin embargo prácticamente la misma duración que la variable global dato, pues no desaparece hasta que no termina la función main.)

Ejemplo 2. Casos particulares.

#include <stdio.h>
int dato1;
static int dato2;
void HacerAlgo(void)
{
static int k = 0;
int j = 0;
k++; j++; dato1 = 10;
printf("H:%d,%d,%d\n", k,j,dato1)
}
	void main()
	{
	int valor = 5;
	dato1 = valor;
	HacerAlgo();
	dato2 = dato1
	HacerAlgo();
	printf("M:%d",dato2);
	}
		El programa mostraría
		H:1,1,10
		H:2,1,10
		M:10

Variables globales: dato1 y dato2.
Variables locales : k y j de la función HacerAlgo y valor de la función main.
Variable estáticas: dato1, dato2 (globales) y k (local de la función HacerAlgo)
Variables automáticas: j y valor.(Esta variable tendría sin embargo prácticamente la misma duración que la variable global dato1, dato2, y la variable local k)

La variable local k al ser estática mantiene el valor anterior entre cada llamada. La variable global dato2 al ser estática sólo podría ser accesible por funciones que estuvieran dentro del mismo fichero fuente (módulo)