Unidad 1 - Conceptos de C

Introducción.

El C es un lenguaje de programación de propósito general. Sus principales características son:

- Programación estructurada.
- Economía en las expresiones.
- Abundancia en operadores y tipos de datos.
- Codificación en alto y bajo nivel simultáneamente.
- Reemplaza ventajosamente la programación en ensamblador.
- Utilización natural de las funciones primitivas del sistema.
- No está orientado a ningún área en especial.
- Producción de código objeto altamente optimizado.
- Facilidad de aprendizaje

El lenguaje C nació en los laboratorios de la Bell Telephone y ha sido estrechamente asociado con el sistema operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador C y la casi totalidad de los programas y herramientas de UNIX, fueron escritos en C. Su eficiencia y claridad han hecho que el lenguaje ensamblador apenas haya sido utilizado en UNIX.

Este lenguaje está inspirado en el lenguaje B escrito por Ken Thompson en 1970 con intención de re codificar el UNIX, que en la fase de arranque estaba escrito en ensamblador, en vistas a su transportabilidad a otras máquinas. B era un lenguaje evolucionado e independiente de la máquina, inspirado en el lenguaje BCPL concebido por Martín Richard en 1967.

En 1972, Oennis Ritchie, toma el relevo y modifica el lenguaje B, creando el lenguaje C y reescribiendo el UNIX en dicho lenguaje. La novedad que proporciono el lenguaje C sobre el B fue el diseño de tipos y estructuras de datos.

Los tipos básicos de datos eran **int** (entero), **char** (carácter), **float** (reales en simple precisión) y **double** (reales de doble precisión). Posteriormente se añadieron los tipos **long**, **short**, y **unsigned** (enteros sin signo). Los tipos estructurados básicos de C son las estructuras, las uniones y las enumeraciones. Estos permiten la definición y declaración de tipos derivados de mayor complejidad.

Las instrucciones de control de flujo de C son las habituales de la programación estructurada: **if**, **for**, **while**, **switch-case** todas incluidas en su predecesor BCPL.

C incluye también punteros y funciones. Los argumentos de las funciones se pasan por valor, esto es copiando su valor, lo cual hace que no se modifiquen los valores de los argumentos en la llamada. Cuando se desea modificar los argumentos en la llamada, estos se pasan por referencia, esto es se pasan las direcciones de los argumentos. Por otra parte, cualquier función puede ser llamada recursivamente.

Una de las peculiaridades de C es su riqueza de operadores. Puede decirse que prácticamente dispone de un operador para cada una de las posibles operaciones en código máquina.

Hay toda una serie de operaciones que pueden hacerse con el lenguaje C que realmente no están incluidas en el compilador propiamente dicho, sino que las realiza un preprocesador justo antes de cada compilación. Las dos más importantes son #**define** (directriz de sustitución simbólica o de definición) #**include** (directriz de inclusión en el fichero fuente).

Finalmente, C, que ha sido pensado para ser altamente transportable y para programar lo improgramable, igual que otros lenguajes tiene sus inconvenientes.

Carece de instrucciones de entrada/salida, de instrucciones para manejo de cadenas de caracteres, con lo que este trabajo queda para la librería de rutinas, con la consiguiente pérdida de transportabilidad. La excesiva libertad en la escritura de los programas puede llevar a errores en la

programación que, por ser correctos sintácticamente no se detectan a simple vista. Por otra parte las precedencias de los operadores convierten a veces las expresiones en pequeños rompecabezas. A pesar de todo. C ha demostrado ser un lenguaje extremadamente eficaz y expresivo.

Sintaxis General del Lenguaje.

El código de un programa C/C++ se guarda en lo que se llama módulos de código que no son mas que archivos de texto que tienen generalmente la extensión ".C " o ".CPP" según sea un programa en lenguaje C o C++ respectivamente. Estos módulos de código tienen una estructura general siempre idéntica, la cual es una colección de cualquier número de directrices para el compilador, declaraciones, definiciones, expresiones, sentencias y funciones.

Todo programa C debe tener una función main(), donde el programa comienza a ejecutarse, las llaves ({}) que incluyen el cuerpo de esta función principal indican el principio y fin del programa.

Un programa C, además de la función main principal, consta de otras funciones que definen rutinas con una función específica dentro del programa.

Para simplificar la estructura de estos módulos veamos un programa ejemplo sencillo:

El ejemplo consiste en pasar de grados Centígrados a Fahrenheit.

```
/* ºF=9/5*C+32 */                    // "/*…*/"  Operadores para realizar comentarios

/* Directivas para el compilador # */

#include "stdio.h"                   // /* cabecera estándar de C*/
#include “conio.h”

/* Definiciones de constantes */

#define INF -30                      // directiva que define a INF como -30
#define SUP 100                      // directiva que define a SUP como 100

/* Definiciones de variables globales */

float fahrenheit;

/* Declaración de funciones */

float conversion(int c);

int main()                                   // función principal, comienza el programa
{
        /* Declaración de variables locales */

        int centigrados;
        int incremento = 6;          // declaración e inicialización

        centigrados=INF;             // asignación
```

```
        while(centigrados <= SUP){
                /* se llama a la función y se le pasa un valor */

                fahrenheit = conversion(centigrados);

                printf("%10d C %10.2f F\n", centigrados, fahrenheit);
                centigrados += incremento;
        }

        getch();
        return 0;
}                                       // fin función principal

float conversion(int cent)              // cabecera de función
{
        float fahr;                     // variable local de la función

        fahr=9.0/5.0*cent+32;
        return fahr;                    // retorna el valor a la sentencia llamada
}                                       // fin de la función conversión
```

Tipos de datos.

El lenguaje C soporta los siguientes tipos de datos:
Tipos de enteros: **char**, **int**, **short**, **long**, **signed**, **unsigned.**
Tipos reales: **float**, **double** y **long double.**
Otros tipos: pointers, arrays y structures.

Tipo	Tamaño [Byte]	Valores que puede contener
char	1	Enteros comprendidos entre -128 y 127
unsigned char	1	Enteros comprendidos entre 0 y 255
int	4	Enteros comprendidos entre -2147483648 y 2147483647
unsigned int	4	Enteros entre 0 y 4294967195
short int	2	Enteros entre -32768 y 32767
long	4	Igual que int
unsigned long	4	Igual que unsigned int
float	4	Reales de simple precisión. Tienen siete dígitos de precisión. Pueden estar comprendidos entre $\pm 3.4e^{38}$ a $\pm 3.4e^{-38}$
double	8	Reales en doble precisión. Tienen hasta 15 dígitos significativos. Están comprendidos entre $\pm 1.7e^{308}$ a $\pm 1.7e^{-308}$
long double	10, 12, 16	Reales de double precisión con 18 dígitos significativos (procesadores de 32 bits)

		comprendidos entre $\pm 1.1e^{4932}$ a $\pm 3.4e^{-4932}$

Tipos adicionales de datos.

enum

Se utiliza para definir una enumeración, que es un conjunto definido de constantes enteras. Comúnmente declaramos una enumeración de la siguiente manera:

enum nombre {lista};

Ejemplo:
enum diahabil {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES};

Este ejemplo define un tipo enumerado cuyas constantes enumeradas son LUNES, MARTES, MIERCOLES, JUEVES, y VIERNES. El valor de la primer constante es por defecto 0, a menos que se le especifique algún valor. Ejemplo:

enum diahabil {LUNES = 10, MARTES, MIERCOLES = 100, JUEVES, VIERNES};

En este caso LUNES toma el valor 10, y MIERCOLES el valor 100. Las constantes que no se le asignó algún valor en la enumeración toman el valor de la constante anterior sumado 1, por lo tanto MARTES será 11, JUEVES será 101, y VIERNES será 102.

Ejemplo:

```
#include <stdio.h>

int main(void)
{
        int proximo_dia;
        enum diahabil {LUNES = 1, MARTES, MIERCOLES, JUEVES, VIERNES} dia;

        printf("Ingrese DIA [1-5]: ");
        scanf("%d", &dia);

        if(dia == LUNES)
                proximo_dia = MARTES;
        else
                proximo_dia = dia+1;
        printf("Proximo DIA = %d\n", proximo_dia);
        return 0;
}
```

pointers (punteros).

Un puntero es una dirección de memoria donde se localiza un objeto de un tipo especificado. Para definir en tipo puntero se utiliza el modificador *****.

Ejemplo: **int** *p;
char *plineas[40];

en este ejemplo se declara un puntero p a un valor entero y un array de punteros plineas (plineas[0] a plineas[39]) a valores de tipo char.

struct

El tipo **struct** se utiliza para declarar variables que representan registros, formado por uno o más campos de diferente o igual tipo.

Ejemplo:

```
struct{
        float real, imag;
        } complejo;

struct persona{
        char nombre[20];
        char apellidos[40]
        long dni;
        }
struct persona reg;
```

En este ejemplo se declara a las variables **complejo** y **reg** como estructuras o registros.

union

Sirve para definir uniones que tienen las mismas formas que las estructuras. Las uniones a diferencia de las estructuras representan registros variables. Esto quiere decir que una variable de este tipo puede alternar entre varios tipos.

arrays

Un array es un conjunto de objetos todos del mismo tipo que ocupan posiciones sucesivas en memoria. Para definir un array se utiliza el modificador **[]**.

Ejemplo:

```
int lista[40];
```

void

El tipo void se utiliza para declararar funciones que no retornan ningún valor o para declararar punteros a un tipo no especificado. Si void aparece entre paréntesis a continuación del nombre de la función, no es interpretado como un tipo. En este caso indica que la función no acepta valores.

Ejemplo: **double** fx(**void**);

**void ***p;

const

El tipo const es utilizado como un modificador de un tipo fundamental. Indica que el valor de un objeto o de un puntero no puede ser modificado.

volatile

Este tipo es utilizado de la misma forma que el tipo **const**. Indica que un objeto puede ser cambiado por otros procesos diferentes al programa actual.

Declaraciones de Tipos.

typedef

Permite declarar nuevos nombres de tipos de datos (sinónimos).

typedef tipo_C identificador[, identificador]...;

tipo_C es cualquier tipo definido en C incluyendo los tipos función, puntero y arrays.

identificador es el nuevo nombre dado a tipo_C.

Ejemplo: **typedef int** ENTERO;
typedef int (***PFI**)();

Este ejemplo propone el tipo ENTERO como sinónimo de int y el tipo PFI como un puntero a una función que devuelve un valor entero.

De acuerdo con esto las declaraciones:

ENTERO n;
PFI p;

Declaran n como una variable de tipo entero y p como un puntero a una función que devuelve un valor entero.

Las declaraciones **typedef** permiten parametrizar un programa para evitar problemas de portabilidad. Esto es utilizando typedef con los tipos que pueden depender de la instalación, solo se tendrán que cambiar estas declaraciones cuando se lleve el programa a otra instalación.

Constantes

Una constante es un valor que, una vez fijado por el compilador, no cambia durante la ejecución del programa. Una constante en C puede ser un número, carácter o una cadena de caracteres.

Constantes enteras.

El lenguaje C permite especificar un entero en base: 10, 8 y 16.

Ejemplo: 256 especifica el número 256 en decimal
0400 especifica el número 256 en octal
0x100 especifica el número 256 e hexadecimal

Constantes reales.

Una constante real tiene el siguiente formato:

[dígitos][.dígitos][(E o e)[±]dígitos]

Ejemplo:
17.24
.008e3
27e-3
Una constante real tiene siempre un tipo double.

Constante de un solo carácter.

Este tipo de constante esta formado por un único carácter encerrado entre comillas simples.

Ejemplo:
' '
'x'
'\x1B' (carácter ASCII que indica escape)

Constante de caracteres.

Una constante de caracteres es una cadena de caracteres encerrados entre comillas dobles.

Ejemplo:
"Esto es una constante de caracteres"
"3.1415926"

Operadores.

Los operadores son símbolos que indican como son manipulados los datos.

Operadores aritméticos	
Operador	Operación
+	Suma. Los operandos pueden ser enteros o reales.
-	Resta. Los operandos pueden ser enteros o reales.
*	Multiplicación. Los operandos pueden ser enteros

	o reales.
/	División. Los operandos pueden ser enteros o reales. Si ambos operandos son enteros el resultado es entero. En el resto de los casos el resultado es real.
%	Módulo o resto de una división entera. Los operandos tienen que ser enteros.

Operadores lógicos	
Operador	Operación
&&	AND. Da como resultado el valor lógico 1 si ambos operandos son distintos de cero. Si uno de ellos es cero el resultado es el valor lógico O. Si el primer operando es igual a cero, el segundo operando no es evaluado.
||	OR. El resultado es 0 si ambos operandos son 0. Si uno de los operandos tiene un valor distinto de 0, el resultado es 1. Si el primer operando es distinto de cero, el segundo operando no es evaluado.
!	NOT. El resultado es 0 si el operando tiene un valor distinto de cero, y 1 en caso contrario. El resultado es de tipo int. El operando puede ser entero, real o un puntero.

Operadores de relación Los operando pueden ser de tipo entero, real o puntero.	
Operador	Operación
<	Primer operando menor que el segundo.
>	Primer operando mayor que el segundo.
<=	Primer operando menor o igual que el segundo.
>=	Primer operando mayor o igual que el segundo.
==	Primer operando igual que el segundo.
!=	Primer operando distinto del segundo.

Operadores unitarios	
Operador	Operación
-	Cambia de signo al operando. El operando puede ser entero o real.
~	Complemento a 1. El operando tiene que ser entero.

Operadores lógicos para manejo de bits	
Operador	Operación
&	Operación AND a nivel de bits.
|	Operación OR a nivel de bits.
^	Operación XOR a nivel de bits.
<<	Rotación o desplazamiento a la izquierda.

>>	Rotación o desplazamiento a la derecha.

Operadores de asignación	
Operador	Operación
++	Incremento.
--	Decremento.
=	Asignación simple.
*=	Multiplicación más asignación.
/=	División más asignación.
%=	Módulo más asignación.
+=	Suma más asignación.
-=	Resta más asignación.
<<=	Desplazamiento a izquierdas más asignación.
>>=	Desplazamiento a derechas más asignación.
&=	Operación AND sobre bits más asignación.
|=	Operación OR sobre bits más asignación.
^=	Operación XOR sobre bits más asignación.

Otros operadores	
Operador indirección	
*	Accede a un valor a través de un puntero. El resultado es el valor direccionado por el operando.
Operador de dirección-de	
&	Da la dirección de su operando. Este operador no se puede aplicar a un campo de bits perteneciente a una estructura o a un identificador declarado como del tipo register.
Operador tamaño de	
sizeof(expresión)	Da como resultado un entero indicando el tamaño en bytes del identificador o tipo especificado, donde *expresión* es un identificador o tipo básico.

Conversiones de tipo.

Cuando los operando dentro de una expresión son de tipo diferente, se convierten a un tipo común de acuerdo con las reglas que se exponen a continuación.

Las reglas que se exponen, se aplican en ese orden, para cada operación binaria perteneciente a una expresión, siguiendo el orden de evaluación expuesto anteriormente.

1. *Cualquier operando de tipo float es convertido a tipo double.*
2. *Si un operando es de tipo long double, el otro operando es convertido a tipo long double.*
3. *Si un operando es de tipo double, el otro operando es convertido a tipo double.*
4. *Cualquier operando de tipo char o short es convertido a tipo int.*
5. *Cualquier operando de tipo unsigned char o unsigned short es convertido a tipo unsigned int.*

6. Si un operando es de tipo unsigned long, el otro operando es convertido a unsigned long.

7. Si un operando es de tipo long, el otro operando es convertido a tipo long.

8. Si un operando es de tipo unsigned int, el otro operando es convertido a tipo unsigned int.

Entrada y Salida Estándar.

Directriz **#include**.

Las operaciones de entrada y salida no forma parte del conjunto de sentencias de C, sino que pertenecen al conjunto de funciones de librerías estándar de C. Por ello todo modulo fuente que utilice funciones de entrada y salida y en general funciones de la librería estándar deberá contener la línea:

a) #**include** "stdio.h" o b) #**include** <stdio.h>

Con estas directrices se le dice al compilador que incluya el archivo de cabecera stdio.h en el programa fuente. Cuando se usan las dobles comillas el compilador buscará este fichero en el directorio actual de trabajo y si no lo encuentra la seguirá buscando en el directorio estándar para los ficheros (directorio include). Cuando se utilizan <> se le dice al compilador que busque el fichero stdio.h solo en el directorio estándar.

Salida con formato. Función printf.

La función printf escribe con formato una serie de caracteres o un valor, en el fichero de salida estándar stdout. Devuelve el número de caracteres escritos.

int *printf*(formato[, argumento]…);

formato: especifica como va a ser la salida. Está formado por caracteres ordinarios, secuencias de escape y especificaciones de formato. El formato se lee de izquierda a derecha. Cada argumento debe tener su correspondiente especificación y en el mismo orden. Si hay más de un argumento que especificaciones de formato, los argumentos en exceso se ignoran.

argumento: representa el valor o valores a escribir.

Una especificación de formato esta compuesta por:

%[flag][ancho][.precisión][{F/N/h/I/L}]tipo

flag	significado
-	Justifica el resultado a la izquierda, dentro del ancho especificado. Por defecto la justificación se hace a la derecha.
+	Antepone el signo + (mas) o – (menos) al valor de salida. Por

	defecto solo se pone signo menos a los valores negativos.
blanco	Antepone un blanco a valor de salida si es positivo. Si se utiliza junto con + entonces se ignora.
#	Cuando se utiliza con especificaciones de formato o, x, o X, antepone al valor de salida 0, 0x, o =X respectivamente. Cuando se utiliza con la especificación de formato e, E, o f, fuerza a que el valor de salida contenga un punto decimal en todos los casos. Cundo se utiliza con la especificación de formato g, o G, fuerza a que el valor de salida contenga un punto decimal en todos los casos y previene que los ceros arrastrados sean truncados. Se ignora con e, d, i, u, o s.
ancho	Mínimo número de posiciones para la salida. Si el valor a escribir ocupa más posiciones de las especificadas, el ancho es incrementado en lo necesario.

precisión	El significado depende del tipo de salida.
tipo	**Significado** (precisión)
d, i, u, o, x, X	La precisión especifica el mínimo número de dígitos que se tienen que escribir. Si es necesario se rellena con ceros a la izquierda. Si el valor excede de la precisión, no se trunca.
e, E, f	La precisión especifica el número de dígitos que tienen que ser escritos después del punto decimal. El valor es redondeado. Por defecto, es 6.
g, G	La precisión especifica el máximo número de dígitos significativos (6 por defecto) que se tienen que escribir.
c	La precisión no tiene efecto.
s	La precisión especifica el máximo número de caracteres a ser escritos. Los caracteres que excedan se ignoran.

F, N, h, I, L	
F	Utilizado en el modelo small para escribir valores que han sido declarados far.
N	Utilizado en los modelos medium y large para escribir valores que han sido declarados near. F y n no pertenecen al C estándar y solamente se utilizarán con los tipos s y p.
h	se utiliza como prefijo con los tipos d, i, o, x, y X, para especificar que el argumento es sbort int, o con u para especificar un sbort unsigned int.
I	Se utiliza como prefijo con los tipos d, i, o, x, y X, para especificar que el argumento es long int, o con u para especificar un long unsigned int. También se utiliza con los tipos e, E, f, g, y G para especificar un double antes que un float.

L	Se utiliza como prefijo con los tipos e, E, g, y G, para especificar long double.

tipo	
carácter	**salida**
d	(int) enteros con signo en base 10
i	(int) enteros con signo en base 10.
u	(int) enteros sin signo en base 10.
o	(int) enteros sin signo en base 8.
x	(int) enteros sin signo en base 16 (01...abcdef).
X	X (int) enteros sin signo en base 16 (01...ABCDEF).
f	(double) valor con signo de la forma: [-]dddd.dddd. El número de dígitos antes del punto decimal depende de la magnitud del número y el número de decimales de la precisión.
e	(double) valor con signo de la forma [-]d.dddde[±]ddd
E	(double) valor con signo de la forma [-]d.ddddE[±]ddd
g	(double) valor con signo en formato f o e (el que sea más compacto para el valor y precisión dados).
G	(double) igual que g, excepto que G introduce el exponente E en vez de e.
c	(int) un solo carácter.
s	(cadena de caracteres) escribir una cadena de caracteres.
n	(puntero a un entero). En el entero es almacenado el número de caracteres hasta ahora en el buffer.
p	Escribe una dirección segmentada (xxxx.yyyy). Si se especifica %Np se escribe solamente el offset de la dirección. %p espera un puntero a un valor lar, por ello bajo el modelo small, utilizar con el argumento a escribir, el tipo cast: (tipo far *)arg.

Ejemplo:

```
#include "stdio.h"

int main(){
        char car;
        static char nombre[]="La temperatura ambiente";
        int a,b,c; float x,y,z;
        car = 'C'; a = 20; b = 350; c = 1988;
```

```
        x = 34.5; y = 1234; z = 1.248;

        printf("\n%s es de ", nombre);
        printf("\n");
        printf("a=%6d \tb = %6d \tc =%6d \n",a, b, c);
        printf("\nLos resultados son los siguientes:\n");
        printf("\n%5s \t \t%5s \t \t%5s\n","x","y","z");
        printf("-----------------------------------------------\n");
        printf("\n%8.2f\t%8.2f\t%8.2f", x, y, z);
        printf("\n%8.2f\t%8.2f\t%8.2f\n", x+y, y/5, z*2);
        printf("\n\n");
        z *= (x+y);
        printf("Valor resultante: %.3f\n", z);

        return 0;
}
```

Entrada con formato. Función scanf.

La función scanf lee datos de la entrada estándar stdin, los interpreta de acuerdo con el formato indicado y los almacena en los argumentos especificados. Cada argumento debe ser un puntero a una variable cuyo tipo se debe corresponder con el tipo especificado en el formato. La función scanf devuelve un entero correspondiente al número de datos leídos de la entrada. Si este valor es cero, significa que no han sido asignado campos. La función scanf, retorna EOF, cuando se intenta leer un end-of-file.

```
# include < stdio.h >

int scanf(formato[, argumento]...);
const char *formato;
```

formato: interpreta cada dato de entrada. Está formado por caracteres en blanco (' ', \t, \n), caracteres ordinarios, y especificaciones de formato. El formato se lee de izquierda a derecha.
Cada argumento debe tener su correspondiente especificación y en el mismo orden.
Si un carácter en la entrada estándar no se corresponde con la entrada especificada por el formato, se interrumpe la entrada de datos.

argumento: es un puntero a la variable que se quiere leer.

Cuando se especifica más de un argumento, los valores correspondientes en la entrada hay que separarlos por uno o más espacios en blanco (' ', \t, \n) o por el carácter que se especifique en el formato.

Un espacio en blanco antes o después de una especificación de formato hace que scanf lea, pero no almacene, todos los caracteres espacio en blanco hasta encontrar un carácter distinto de espacio en blanco.

Ejemplo: **int** a; **float** b; **char** c;

sentencia	entrada de datos
scanf("%d %f %c", &a, &b, &c);	5 23.4 b
scanf("%d%f%c", &a, &b, &c);	5 23.4 b
scanf("%d,%f,%c", &a, &b, &c);	5,23.4,b
scanf("%d , %f , %c", &a, &b, &c);	5, 23.4, b
scanf("%d : %f : %c", &a, &b, &c);	5:23.4 : b

Una especificación de formato está compuesta por:

%[*][ancho][{F/N}][(h/I}]tipo

Una especificación de formato siempre comienza con %

*	Un asterisco a continuación de símbolo % suprime la asignación del siguiente dato en la entrada. Ejemplo: scanf("%d %*s %d %*s", &horas, &minutos); Entrada: 12 horas 30 minutos Resultado: las cadenas horas y minutos no se asignan
ancho	Máximo número de caracteres a leer de la entrada. Los caracteres en exceso no son tenidos en cuenta.
F	Indica que se quiere leer un valor apuntado por una dirección far (dirección segmentada).
N	Indica que se quiere leer un valor apuntado por una dirección near (dirección dada por el valor offset). F y N no pertenecen al C estándar.
h	Se utiliza como prefijo con los tipos d, i, n, o, y x, para especificar que el argumento es short int, o con u para especificar un short unsigned int.
I	Se utiliza como prefijo con los tipos d, i, n, o, y x, para especifar que el argumento es long int, o con u para especificar un long unsigned int. También se utiliza con los tipos e, f, y g para especificar un double.

tipo		El tipo determina si el dato de entrada es interpretado como un carácter, como una cadena de caracteres o como un número. El formato más simple contiene el símbolo % y el tipo. Por ejemplo: %i
carácter	**el argumento es un puntero a**	**entrada esperada**
d	int	Enteros con signo en base 10.
D	long	Enteros con signo en base 10.
i	int	Entero en base 10,16 u 8.
I	long	Entero en base 10,16 u 8.
u	unsigned int	Enteros sin signo en base 10.
U	unsigned long	Enteros sin signo en base 10.
o	int	Enteros sin signo en base 8.
O	long	Enteros sin signo en base 8.

x	int	Enteros sin signo en base 16.
X	long	Enteros sin signo en base 16.
f, e, E, g, G	double	valor con signo de la forma [-]d.dddde[±ddd]
c	char	Un solo carácter.
s	char	Cadena de caracteres.

n	int	En el entero es almacenado el número de caracteres leídos del buffer o fichero. Por ejemplo:
		long a; int r;
		scanf("%Id%n",&a, &r);
		printf("Caracteres leidos:" %Id\n",r);

p	Puntero a un puntero far o void	Valor de la forma xxxx.yyyy, donde x e y representan lo dígitos del 0 al 9 y letras de la A a la F

Ejemplo:

```
#include <stdio.h>

int main(){
        int a,b,r;
        char c, s[20];

        printf("Introducir un valor entero, un real, un char y una cadena\n = >");
        r = scanf("%d %f %c %s",&a,&b,&c,&s);
        printf("\nNumero de datos leidos: %d\n",r);
        printf("Datos leidos: %d %f %c %s\n",a,b,c,s);
        printf("\n\n");
        printf("Valor hexadecimal: ");
        scanf("%x",&a);
        printf("Valor decimal:      %i\n",a);

        return 0;
}
```

La sentencia scanf lee datos delimitados por espacios en blanco. Entonces para poder leer cadenas de caracteres que contengan espacios en blanco debemos cambiar la especificación %s por % [^\n], por ejemplo que indica leer caracteres hasta encontrar un carácter \n.

Entrada de caracteres – getchar().

Cabecera: stdio.h
Sintaxis:

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

int getchar(void);

Lee un carácter de la entrada estándar stdin y avanza la posición de lectura al siguiente carácter a leer. Si la la función getchar() ocurre satisfactoriamente devuelve el carácter leído luego de convertirlo a un int sin signo. En caso de error o de un end-of-file, la función devuelve un EOF.

Salida de caracteres – putchar.

Cabecera: stdio.h
Sintaxis:

#**include** <stdio.h>
int putchar(int c)

Escribe un carácter a la salida estándar stdout en la posición actual y avanza a la siguiente posición de escritura. Si sucede la función devuelve el carácter c, en caso contrario devuelve un EOF.

Funciones getch y getche.

Cabecera: conio.h
Sintaxis:

#**include** <conio.h>
int getch(void);

Esta función lee un carácter directamente desde teclado sin eco a la pantalla.

La función getche tiene las mismas características que la anterior solo que esta realiza un eco a la pantalla.

Sentencias de Control.

El conjunto de sentencias de control esta compuesta par las sentencias de decisión (if-else, switch-case) y las sentencias de bucle (**while**, **do-while**, **for**).

Sentencia if-else.

Toma una decisión referente a una acción a ejecutar en programa, basándose en el resultado (verdadero o falso) de una expresión.

```
if (expresión)
        sentencia1;
else
        sentencia2;
```

expresión: debe ser una expresión numérica, relacional o lógica. El resultado que se obtiene al evaluarla es verdadero (no cero) o falso (cero).

sentencia1/2: representa una sentencia simple o compleja. Cada sentencia simple debe estar separada de la anterior por un punto y coma.

Si el resultado de la expresión es verdadero se ejecuta la sentencia1, en caso de que este resultado haya sido falso se ejecuta la sentencia2.

En el caso de que la expresión de como resultado falso, y la cláusula else no existe se ignora la sentencia1.

En cualquier caso la ejecución continua con la siguiente sentencia ejecutable.

Ejemplo:

```
if (x)
        b  =  a/x;
b  = b + 1;

if (a == (b * 5)){
        x = 4;
        a = a + x;
}
else
        b = 0;
```

Como se observa en los ejemplos anteriores cuando se debe ejecutar más de una sentencia ejecutable dentro del if se utilizan las llaves ({…}) para indicar donde comienza y termina dicha sentencia que llamaremos compuesta.

Las sentencias pueden estar anidadas. Esto es que como sentencia1 o como sentencia2 aparezca un **if**. Es decir:

```
if (expresión1){
        if (espresión2)
                sentencia1;
}
else
        sentencia2;
```

Ejemplo:

```
#include <stdio.h>

int main()
{
        int a, b;
        a = 10;
        b = 5;

        if (a > b)
                printf("%d es mayor que %d", a, b);
        else{
                if (a < b)
                        printf("%d es menor que %d", a, b);
                else
                        printf("%d es igual que %d", a, b);
        }

        getchar();
        return 0;
}
```

Sentencia switch - case

Esta sentencia permite ejecutar una o varias acciones, en función del valor de una expresión.

```
switch (exp-test){
        declaraciones;
        case cte1:
                sentencia1;
                break;
        case cte2:
                sentencia2;
                break;
        .
        .
        .
        default:
```

```
        sentenciaN;
}
```

exp-test: es una constante entera, una constante de caracteres o una expresión constante. El valor es convertido a tipo int.

sentencia: es una sentencia simple o compuesta.

Al principio de la sentencia switch pueden aparecer declaraciones, si no las hay son ignoradas.

La sentencia switch evalúa la expresión y las compara con cada constante, ejecutando la sentencia que coincida con el resultado de la exp-test. En caso de que no se encuentre el resultado de la exp-test en los case se da la posibilidad para que se ejecute la sentencia por defecto (en este caso sentenciaN). Si la opción default no existe esta se ignora.

Sentencia while.

Ejecuta una sentencia simple o compuesta una o más veces dependiendo del valor de una expresión.

```
while (expresión)
        sentencia;
```

expresión: cualquier expresión numérica, relacional o lógica.

sentencia: es una sentencia simple o compuesta.

La ejecución de la sentencia ocurre de la siguiente forma:

1. Se evalúa la expresión.
2. Si el resultado anterior es cero (falso), la sentencia no se ejecuta y se pasa a ejecutar la siguiente sentencia en el programa.
3. Si el resultado de la expresión es distinto de cero (verdadero), se ejecuta la sentencia del while y el proceso se repite comenzando en el punto 1.

Ejemplo:

```
#include <stdio.h>

int main()
{
        char car;

        car  =  '\0';
        printf("\nDesea continuar s/n (si o no) ");
        while ((car =getche()) != 's' && car != 'n')
                printf("\nDesea continuar s/n (si o no) ");
```

```
        return 0;
}
```

La rutina anterior solicita obligatoriamente de las dos repuestas posibles: s/n (si o no).

Sentencia do-while.

Ejecuta una sentencia simple o compuesta, una o más veces, dependiendo del valor de una expresión.

```
do
        sentencia;
while (expresión);
```

expresión: cualquier expresión numérica, relacional o lógica.

sentencia: es una sentencia simple o compuesta.

La ejecución de esta sentencia sucede de la siguiente forma:

1. Se ejecuta la sentencia o cuerpo de la sentencia do.
2. Se evalúa la expresión.
3. Si el resultado de la evaluación de la expresión es cero (falso), la sentencia no se ejecuta y se pasa a ejecutar la siguiente sentencia en el programa.
4. Si el resultado de la evaluación de la expresión es distinto de cero (verdadero), el proceso se repite comenzando en el punto 1.

Sentencia for.

Cuando se desea ejecutar una sentencia simple o compuesta un número veces conocido, la construcción adecuada es la sentencia for.

```
for (v1=e1, v2=e2,…; condición; progresión-cond)
        sentencia;
```

vi=ei	representa una variable que será inicializada con el valor ei
condición	es una expresión de Boole. Si se omite se supone que siempre es verdadera
progresión-cond	es una expresión cuyo valor evoluciona en el sentido que se de la condición para finalizar la ejecución de la sentencia **for**
sentencia	es una sentencia simple o compuesta

;	separa la inicialización, condición y progresión-cond. Siempre deben de escribirse

La ejecución de la sentencia **for** sucede de la siguiente forma:

1. Se inicializan las variables vi.
2. Se evalúa la expresión de Boole (condición).
   2.1 Si el resultado es verdadero, se ejecuta la sentencia, se evalúa la expresión que da lugar a la progresión de la condición y se vuelve al punto 2.
   2.2 Si el resultado de 2 es falso, la ejecución de la sentencia for se da por finalizada y se continúa en la siguiente sentencia del programa.

Ejemplo:

```
#include <stdio.h>

int main()
{
        int i;

        for (i=1; i <= 100; i++)
                printf("%d",i);

        getchar();
        return 0;
}
```

Este ejemplo imprime los números del 1 al 100. Literalmente dice: desde i igual a 1, mientras i sea menor o igual que 100, con incremento 1, escribir el valor de i.

```
#include <stdio.h>

int main()
{
        for (;;)
                printf("%d",i);
}
```

Este ejemplo realiza un bucle infinito.

Problemas Unidad 1.

Uso de funciones I/O:

1. Explique cual es la salida del siguiente programa.

```
#include <stdio.h>

int main(void)
{
        int i = 100;

        i = i+i;
        printf("V1:%d V2:%d\n", i+i, i);
        return 0;
}
```

2. Explique cual es la salida del siguiente programa.

```
#include <stdio.h>
int main(void)
{
        int i = 30;
        float j = 10.65;

        printf("%f %d\n", i, j);
        return 0;
}
```

3. Escribir un programa que declare dos enteros y le asigne los valores 5 y 2. Luego mostrar su suma, diferencia, producto, la división exacta (2.5), y el resto de la división entera.
4. Realizar un programa que nos de cómo resultado el interés producido y el capital total acumulado de una cantidad c, invertida a un interés r al año. (capital = c + intereses, intereses = c*r/100).
5. Un depósito semiesférico de radio R tiene un orificio en el fondo de radio r. el tiempo que tarda en escurrir su contenido líquido es: $t = \frac{14}{15} \cdot \frac{R^{5/2}}{r^2 \cdot \sqrt{2 \cdot G}}$, con G = 9,81. Con los radios R=1.2 y r=0.03 calcular el tiempo, en segundos.
6. Se desea calcular la superficie de un triángulo de base B y altura H, para ello realice un programa que tome los datos necesarios, y los presente en pantalla junto con el área.
7. Realizar un programa que lea la cantidad de segundos del día y los transforme en hh:mm:ss. (Un día tiene 86400seg).

Uso de **if**, **switch**:

8. Realizar un programa que de cómo resultado el menor de tres números diferentes.

9. Al efectuar una compra en un cierto almacén, si adquirimos de un mismo artículo más de 100 unidades, nos hacen un descuento del 40%, entre 25 y 100 un 20%, entre 10 y 24 10%, y no hay descuento para una adquisición de memos de 10 unidades. Calcular el importe a pagar. Ingresar como datos el código del artículo, precio unitario y cantidad comprada.
10. Leer una fecha representada por dos enteros, mes y año y dar como resultado los días correspondientes al mes. Tener en cuenta que febrero puede tener 29 días si el año es bisiesto.
11. Realice un programa que de él importe a pagar por un vehículo al circular por una autopista. Los vehículos son una moto la cual paga $100 por recorrer cualquier cantidad de Km, una bicicleta que no paga, un auto que paga $30 por cada Km recorrido, y un camión que paga $30 por Km más $25 por toneladas. (Utilice una enumeración para el tipo de vehículo).
12. Que resultado da el siguiente programa.

```
#include <stdio.h>
int main(void)
{
        printf("%d\n", ~(~0 << 4));
        return 0;
}
```

13. Realizar un programa que determine si un número es impar. Use operadores Bitwise.
14. Realizar un programa que determine el estado de un bit de un número entero. Para esto ingrese un valor entero y el número de bit que desea evaluar. (Un valor entero tiene 32 bits). Use operadores Bitwise.
15. Realizar un programa que lea un byte y cambie los cuatro bits mas altos por los cuatro bits mas bajos. Use operadores Bitwise.

Uso de **while**:

16. Realizar un programa que simule una máquina sumadora. La entrada de datos finaliza cuando se presiona la tecla F6 (^Z).
17. Realizar un programa que nos imprima los números z, comprendidos entre 1 y 50, que cumplan la expresión: $z^2 = x^2 + y^2$, donde z, x e y son números positivos.

Uso de **for**, **do**:

18. Calcular la raíz cuadrada de un número n, por el método de Newton que dice:
    $r_{i+1}=(n/r_i+r_i)/2$
    la raíz cuadrada será válida, cuando se cumpla que:
    $abs(r_i-r_{i+1})<=épsilon$
19. Escribir un programa que imprima un triángulo construido con caracteres consecutivos del código ASII, como el que se muestra a continuación.

```
!
"   #
$   %   &
'   (   )   *
+   ,   _   .   /
0   1   2   3   4   5
```

20. Imprimir un tablero de ajedrez y sobre el marcar con * las celdas a las que se puede mover un alfil desde un a posición dada.

Ejercicios complementarios:

21. Realizar un programa que calcule la raíz cuadrada de la ecuación:

$$ax^2 + bx + c = 0$$

teniendo en cuenta los siguientes casos:

Si a=0 y b=0 imprimiremos un mensaje diciendo que la ecuación es degenerada.
Si a=0 y b!=0 existe una raíz única con valor –c/b.

En los demás casos usaremos la fórmula siguiente de la cuadrática $r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$

La expresión $d = b^2 - 4 \cdot a \cdot c$ se denomina discriminante.
Si d >= 0 entonces hay dos raíces reales.
Si d < 0 entonces hay dos raíces complejas de la forma:

$$x + yi,\ x - yi$$

Indicar, con literales apropiados los datos a introducir, así como los resultados obtenidos.

22. Realizar un programa que lea números enteros de forma continua hasta que se ingrese 0. Luego reporte cual fue el valor más negativo y el mayor positivo. En el caso de que no hayan ingresado valores negativos o positivos se deberá informar tal situación.
23. Escribir un programa que lea un texto y nos de cómo resultado el número de palabras con al menos cuatro vocales diferentes. Suponemos que una palabra este separada de la otra por uno o más espacios (' '), caracteres tab (\t) o caracteres nueva línea (\n).
24. Escribir un programa que lea un texto y nos de cómo resultado el número de caracteres, el número de palabras y el número de líneas del mismo. Suponemos que una palabra esta separada de la otra por uno o más espacios, caracteres tab o caracteres nueva línea.
25. Realizar un programa que permita represente un número entero en su correspondiente binario.