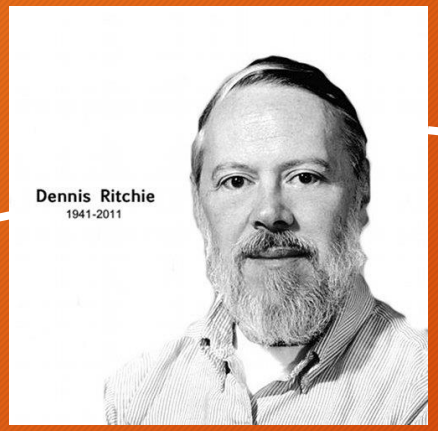
Introducción al Lenguaje

PROGRAMACIÓN I
Otoño 2022
NRC: 10844

Dr. Mario Rossainz López
Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla

Historia y Desarrollo del Lenguaje



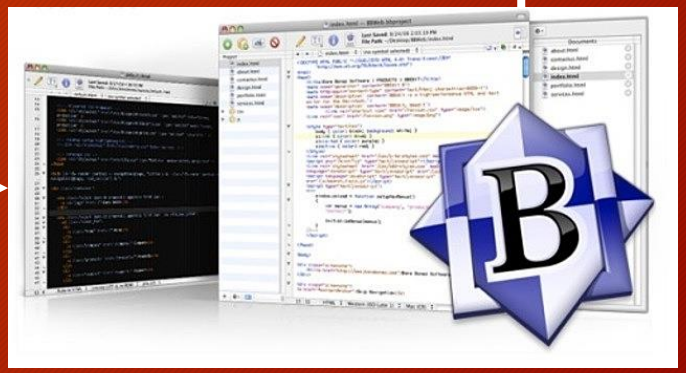
1972

bcpl))


Ritchie and Thompson(sitting) at PDP-11 in Bell Labs

1966

1969



Historia y Desarrollo del Lenguaje

Lenguaje
ANSI C

Lenguaje Nativo

POSIX

- VERSIONES NO ESTANDAR:
- Microsoft C
- Borland C

ANSI = American National Estándar Interface
POSIX= Portable Operating Sistem Interface

Historia y Desarrollo del Lenguaje

CARACTERÍSTICAS DEL LENGUAJE C

- Lenguaje de uso general, con una sintaxis corta y un juego de operadores potente
- Lenguaje de nivel medio. Explota recursos de harware y manejo de direcciones y bits
- No posee operaciones de entrada-salida, ni métodos de archivos y tampoco maneja cadenas de caracteres. Estas operaciones se hacen por medio de funciones contenidas en librerías externas al lenguaje
- Lenguaje estructurado. Utiliza el concepto de función la cual puede tener sus propias variables locales, acceder a variables globales del programa y devolver valores o ninguno.
- C permite una compilación separada de los programas y enlazarlos entre si junto con las bibliotecas externas para formar el programa ejecutable completo.
- C esta definido bajo el estándar ANSI. Su creación y uso sigue las normas de este estándar

Identificadores Estandar

- Son nombres que se les da a las variables, constantes, funciones, tipos y etiquetas.
- Un identificador se forma de letras y dígitos en secuencia, las letras pueden ser mayúsculas o minúsculas (una letra mayúscula es diferente a la misma letra, pero en minúsculas) y los dígitos van del 0 al 9.
- Además se puede utilizar el carácter especial de subrayado o guión bajo (_).
- Como regla, el primer carácter de un identificador debe ser una letra o el carácter de subrayado.

Identificadores Estandar

- Ejemplos:
  - Suma
  - suma
  - Calculo_num_primos
  - Abc123
  - ab12C3
  - _ordenar
  - i

Palabras Reservadas del Lenguaje

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

Estructura de un Programa en C

Directiva

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                    /* inicia El cuerpo del prog. Principal */
     printf("Hola Mundo!\n");   /* imprime en pantalla */

    return;          /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

Cabecera
Header

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                        /* inicia El cuerpo del prog. Principal */
     printf("Hola Mundo!\n");   /* imprime en pantalla */

    return;              /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

Función que se encuentra dentro del header <stdio.h>

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                   /* inicia El cuerpo del prog. Principal */
    printf("Hola Mundo!\n");   /* imprime en pantalla */

    return;          /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

Programa Principal o Función Principal

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
    {                        /* inicia El cuerpo del prog. Principal */
      printf("Hola Mundo!\n");   /* imprime en pantalla */

     return;             /* regresa um valor al sistema */
    }
```

HolaMundo.c

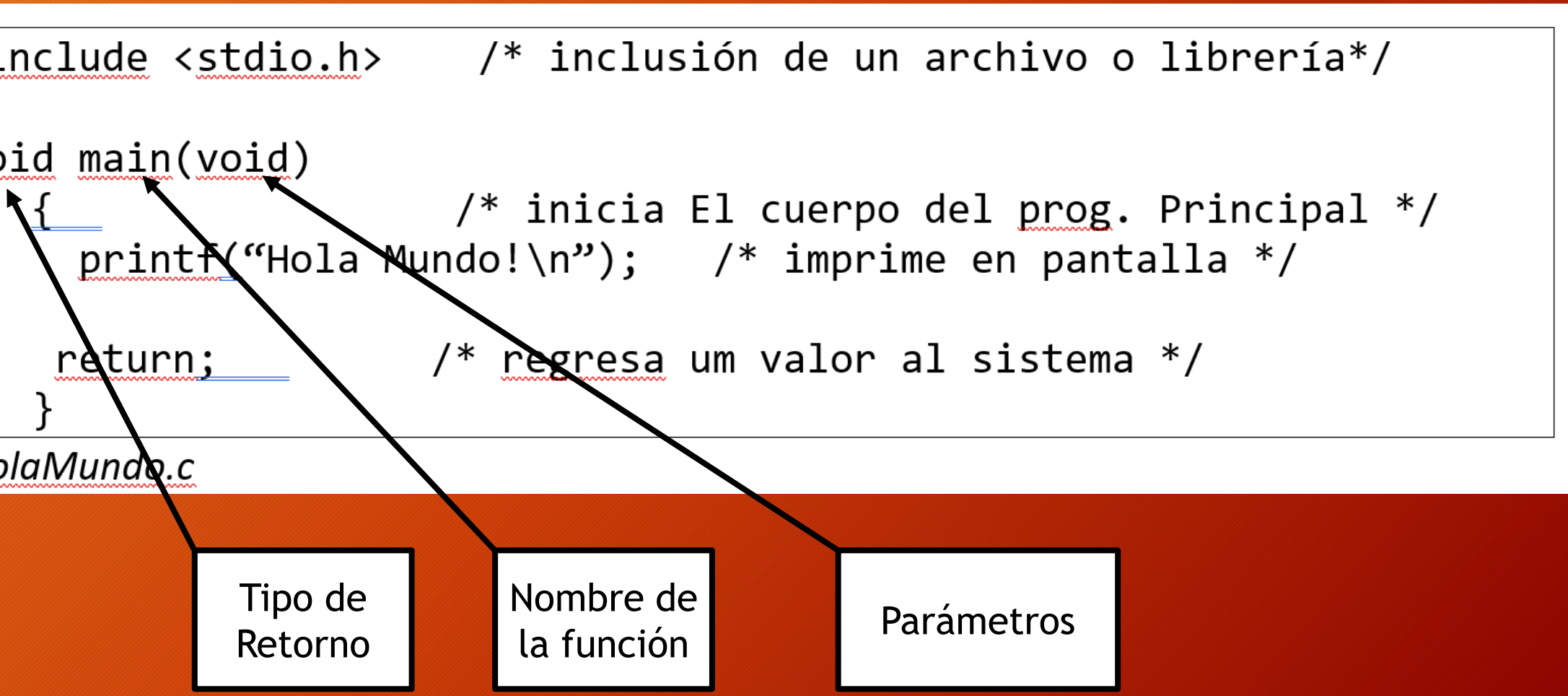
Estructura de un Programa en C

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                          /* inicia El cuerpo del prog. Principal */
     printf("Hola Mundo!\n");   /* imprime en pantalla */

     return;                /* regresa um valor al sistema */
    }
```

HolaMundo.c



Estructura de un Programa en C

Cuerpo de la Función

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                        /* inicia El cuerpo del prog. Principal */
     printf("Hola Mundo!\n");    /* imprime en pantalla */

    return;             /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

Función que imprime una cadena a pantalla

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                        /* inicia El cuerpo del prog. Principal */
    printf("Hola Mundo!\n");   /* imprime en pantalla */

    return;          /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

Palabra reservada para el retorno del valor de una función

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                        /* inicia El cuerpo del prog. Principal */
     printf("Hola Mundo!\n");   /* imprime en pantalla */

    return;              /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

Comentario

```
#include <stdio.h>     /* inclusión de un archivo o librería*/

void main(void)
   {                        /* inicia El cuerpo del prog. Principal */
     printf("Hola Mundo!\n");   /* imprime en pantalla */

    return;          /* regresa um valor al sistema */
   }
```

HolaMundo.c

Estructura de un Programa en C

- Archivos de cabecera (HEADERS)

```
#include <stdio.h>          /*  Directorio de instalación */
#include “stdio.h”          /*  Directorio activo en el momento */
#include “c:\mi_header.h”   /*  Directorio indicado */
```

Tipos de Datos Estandar (Primitivos)

- char
- int
- float
- double
- void

tipo	bytes	v. mínimo	v. máximo
char	1	-127	+127
int	2	-32.767	+32.767
signed char	1	-127	+127
signed int	2	-32.768	+32.767
unsigned char	1	0	+255
unsigned int	2	0	65.535
short int	2	-32.768	+32.767

tipo	bytes	v.mínimo	v. máximo
long int	4	-2.147.483.648	+2.147.483.647
signed long	4	-2.147.483.648	+2.147.483.647
unsigned long	4	0	4.294.967.295
float	4	3.4e-38	3.4e+38
double	8	1.7e-308	1.7e+308
long double	10	3.4e-4932	3.4e+4932

Variables en C

- Una variable es un identificador o posición de memoria representada mediante un nombre que se usa para mantener un valor que puede ser modificado por el programa. Todas las variables en C deben ser declaradas antes de ser utilizadas y su sintaxis es la siguiente:

```
tipo nombre_variable [, nombre_variable, . . ., nombre_variable];
```

- `int i,j,k;`
- `float largo, ancho;`
- `char c;`

Variables en C

- Objetivos de la declaración de una variable:

1. El compilador reserva la cantidad de memoria necesaria para almacenar los valores asociados con las variables.

2. Debido a que se especifican los tipos de datos asociados con las variables, éstas permiten al compilador instruir a la máquina para que desempeñe correctamente ciertas operaciones.

Ámbito de las Variables

Según el lugar donde se declaren las variables éstas tendrán un determinado ámbito (alcance):

- Variables locales
- Variables Globales
- Parámetros Formales

Ámbito de las Variables

VARIABLES LOCALES:

Son aquellas que se declaran dentro de una función y sólo pueden ser referenciadas (utilizadas) instrucciones que estén dentro de la función en que han sido declaradas. No son conocidas fuera de la función. Pierden su valor cuando se sale y se entra en la función. Una variable local también es aquella que está siendo referenciada sólo por sentencias que estén dentro del bloque donde están declaradas. Un bloque es aquel que está encerrado entre dos llaves.

```
void func1(void)      //la variable x declarada en func1(…)
{                     //no es la misma que la declarada en func2(…)
 int x=10;
}

void func2(void)     //La variable "y" declarada en func2(…)
{                    //se crea solo al entrar en el fragmento
int x=-20;           //del programa y desaparece al salir de él
   {
    int y;
    y=2;
   }
}
```

Ámbito de las Variables

VARIABLES GLOBALES:

Son conocidas a lo largo de todo el programa. Se pueden usar en cualquier parte del código. Mantienen sus valores durante toda la ejecución. Las variables globales se declaran por lo general al principio del programa, fuera de todas las funciones, incluida la función *main()*. Inicialmente toman el valor cero o nulo según el tipo.

```
int x;                  //la variable x es global, pues está
                        //declarada fuera de todas
void fun1(…)            // las funciones
{
y=x;                    // en la function fun1(…) se accede a la
x=10;                   //variable global x
}

void fun2(…)            // en la function fun2(…), x se refiere a
{                       // la variable local
 int x;
 x=3;
}
```

Ámbito de las Variables

PARÁMETROS FORMALES:

Son los argumentos de una función. La declaración de éstas variables se hace entre los paréntesis de una función. Su comportamiento es igual que las variables locales de cualquier función, es decir, desaparecen al terminar su ejecución.

```
float mul(float x, float y)  // Los parámetros formales de
{                            // la function mul() son x e y
  return (x*y);
}
```

Constantes

- Una constante es un valor fijo que no puede ser modificado por el programa.
- Puede ser de cualquier tipo de datos básicos.
- Las constantes de carácter van encerradas entre comillas simples, Las constantes enteras se especifican con números sin parte decimal y las de punto flotante con su parte entera separada por un punto de su parte decimal.
- Las cadenas de caracteres irán encerradas entre comillas dobles.

Constantes

Flotantes	Enteros	Cadenas	Caracter
3.10	10	“Hola Mundo”	‘a’
0.987	-1234	“”	‘#’

- SINTAXIS:
- #define identificador valor

```
#define     entero      10
#define     real        1.09982
#define     cad         “Esto es una cadena”
#define     car         ‘a’
```

Expresiones, Proposiciones y Asignaciones

- Una **expresión** es una combinacion de contantes, variables, operadores y llamadas a funciones.
  - tan(1.8)
  - a+b*3.0*9.3242
  - 3.77+sen(3.14*98.7)
- Un **operador** es un símbolo que indica al compilador que se lleven a cabo operaciones específicas. En C se tienen tres clases de operadores: aritméticos, relaciones y lógicos.

- Una **proposición** se forma cuando una expresión que termina con un punto y coma es asignada a una variable. Las proposiciones de asignación tienen la siguiente sintaxis:

variable = expresión;

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 la división entera. Los operandos tienen que ser enteros.
-(unario)	Menos unario. Los operadores pueden ser enteros o reales.

Operadores	Asociatividad
-(unuario)	derecha a izquierda
*,/,%	izquierda a derecha
+,-	izquierda a derecha
=	derecha a izquierda

Operadores Relacionales

Operador	Operación	Ejemplos
<	Primer operando menor que el segundo	a<3
>	Primer operando mayor que el segundo	b>w
<=	Primer operando menor o igual que el segundo	-7.7<=-99.335
>=	Primer operando mayor o igual que el segundo	-1.3>=(2.0*x+3.3)
= =	Primer operando igual que el segundo	c= ='w'
!=	Primer operando distinto del segundo	x!=-2.77

Operadores Lógicos

Operador	Operación	Ejemplo
&&	AND. Da como resultado el valor lógico 1 si ambos operandos son distintos de 0. Si uno de ellos es cero el resultado es el valor lógico 0. Si el primer operando es igual a cero, el segundo operando no es evaluado.	(z<x) && (y>w)
||	OR. El resultado es cero 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 0, el segundo operando no es evaluado.	(x= =y) || (z!=p)
!	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 apuntador.	!a

Operadores de Asignación

Operador	Operación
++	Incremento
--	Decremento
=	Asignación simple
+=	Suma más asignación
-=	Resta más asignación
|=	Operación OR sobre bits más asignación
&=	Operación AND sobre bits más asignación
>>=	Corrimientos a la derecha más asignación
<<=	Corrimientos a la izquierda más asignación
*=	Multiplicación más asignación
/=	División más asignación
%=	Módulo más asignación

Prioridades de los Operadores

Operadores	Asociatividad
( )	Izquierda a derecha
-(unario), ++, --, ~, !, *, &(los últimos dos como operadores apuntadores)	Derecha a izquierda
*,/,%	Izquierda a derecha
+,-	Izquierda a derecha
<,<=,>,>=	Izquierda a derecha
= =, !=	Izquierda a derecha
&	Izquierda a derecha
^	Izquierda a derecha
|	Izquierda a derecha
&&	Izquierda a derecha
||	Izquierda a derecha
=, +=, -=, *=, <<=, >>=, %=	Derecha a izquierda
, (operador coma)	Izquierda a derecha

Operadores para manejo de bits

Operador	Operación
~	Complemento a 1. El operando tiene que ser entero
&	AND a nivel de bits
|	OR a nivel de bits
^	XOR a nivel de bits
<<	Corrimiento (desplazamiento) a la izquierda
>>	Corrimiento (desplazamiento) a la derecha

Operadores de dirección e indirección

- El operador de **dirección (&)** hace referencia a una dirección de memoria

- El operador de **indirección (*)** accede al valor almacenado en una localidad de memoria direccionada por el operador de dirección.

Entrada y Salida básica en C

- Las operaciones de entrada y salida no forman parte del conjunto de sentencias de C.
- Las operaciones de entrada y salida pertenecen al conjunto de funciones de la librería estándar I/O de C (***stdio.h***).
- Cuando se emplea alguna función de entrada y salida, se debe incluir éste archivo al comienzo del programa mediante la directiva: **#include <stdio.h>** y el compilador inserta el código fuente al comienzo del archivo de nuestro programa.
- Las siguientes funciones son algunas de las más utilizadas para la entrada y salida de datos en C: **printf**, **scanf**, **getch**, **getchar**, **puts**, **gets**. Todas y cada una de ellas tiene una sintaxis que las identifica.

Salida con formato [printf( )]

- Para visualizar los datos por la pantalla se dispone de la función ***printf(…)*** que permite formatear la salida. Esta función puede visualizar una combinación de valores numéricos, caracteres y cadenas de caracteres. Su sintaxis es la siguiente:

```
printf(cadena_de_control, arg1, arg2, . . ., argn);
```

- La ***cadena de control*** especifica cómo va a ser la salida. Es una cadena delimitada por comillas, por caracteres ordinarios, secuencias de escape y especificaciones de formato bajo el cual se requiere la salida de la información hacia pantalla.

- La lista de argumentos **arg1**, **arg2**, **. . . argn** representa el valor o valores a escribir en la pantalla.

Salida con formato [printf( )]

ESPECIFICACIONES DE FORMATO:

- Cada uno de los datos que se desee mandar a imprimir debe ir antecedido por el carácter % y después debe de venir (en ese orden) lo siguiente (no es necesario poner todo, lo que se encuentra entre corchetes es opcional):

```
%[flags][width][.prec][F|N|h|l|L] tipo_de_dato
```

Salida con formato [printf( )]

```
%[flags][width][.prec][F|N|h|l|L] tipo_de_dato
```

COMPONENTE	ESPECIFICACIÓN
flags	Justificación, etc.
width	Número de dígitos significativos parte entera
.prec	Número de dígitos significativos parte real
F|N|h|l|L	Modificadores de Salida N= Near apuntador F= far Apuntador h= entero corto l= entero largo L= real largo
Tipo_de_dato	c= imprime un carácter d= imprime un entero e= notación científica s= imprime una cadena f= decimal en punto flotante

Salida con formato [printf( )]

SECUENCIAS DE ESCAPE

Secuencia	Nombre
\ n	Nueva línea
\ t	Tab horizontal
\ v	Tab vertical (solo para impresora)
\ b	Backspace (retroceso)
\ r	Retorno de carro
\ f	Alineación de página (solo para impresora)
\ a	Bell (sonido)
\ ´	Comilla simple
\ ”	Comilla doble
\ 0	Nulo
\ \	Backslash (barra hacia atrás)
\ ddd	Carácter ASCII. Representación Octal
\ xdd	Carácter ASCII. Representación hexadecimal

Salida con formato [printf( )]

EJEMPLOS:

```
char ch = 'A';
int m = 4;
float x = 14.250;

printf("ch=%c, m=%d", ch, m);
printf("ch=%d, x=%f", ch, x);
printf("ch=%x,\n m=%d ", ch, m);

printf("x=%4.1f, m=%05d", x, m);
printf("La suma x+m=%f",(x+m));
printf("La tecla es %c\n",getch());
```

```
El programa produce
las siguientes salidas.


ch = A , m= 4
ch = 65, real = 14.250
ch = 41,
m = 4   (N.B salto de linea).
x = 14.2, m = 00004
Argumento como expresión(x+m)
El argumento es una función.
```

Entrada con formato [scanf( )]

- La función *scanf(…)* lee los datos desde el teclado con formato. Su sintaxis es:

```
scanf(cadena_de_control, arg1, arg2, . . ., argn);
```

- La **cadena de control** contiene información sobre el formato de las entradas y son idénticos a los formatos vistos en la función *printf(…)*. Es decir, se forma por códigos de formato de entrada, que están precedidos por un signo de % y encerrados entre comillas.

Entrada con formato [scanf( )]

Códigos de Formato

Código	Significado
c	Lee un único carácter
d	Lee un entero decimal base 10
i	Lee un entero en base 10, 16 u 8
f	Lee un número en punto flotante
e	Lee un número en punto flotante
h	Lee un número entero corto
s	Lee una cadena de caracteres
o	Lee un entero en octal
x	Lee un número hexadecimal

Entrada con formato [scanf( )]

```
scanf(cadena_de_control, arg1, arg2, . . ., argn);
```

- Los argumentos ***arg1***, ***arg2***, ***arg3***,. . ., ***argn*** son direcciones de las variables y para ellos se debe utilizar el operador de direcciones (&). La lista de valores por tanto representa el valor o valores a escribir en pantalla. Una especificación de formato para scanf() esta compuesta por:

Símbolo	Significado
*	Un asterisco a continuación de % suprime la asignación del siguiente dato en la entrada.
Ancho	Máximo numero 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 en el argumento es short int, o con u para especificar un short unsigned int.
L	Se utiliza como prefijo con los tipos d,i,n,o, y x, para especificar que el argumento es long int. También se utiliza con los tipos e,f y g para especificar un double.

Entrada con formato [scanf( )]

EJEMPLOS:

```
printf("Da un numero: ");
scanf("%d",&n);
```

```
main()
{
int a,r;
float b;
char c;
printf("Introducir un valor entero, un real y un caracter\n =>");
r=scanf("%d%f%c\n",&a,&b,&c);
printf("Numero de datos leidos: %d\n",r);
printf ("datos leidos: %d%f%c\n",a,b,c);
}
```

Entrada con formato [scanf( )]

Ejemplo para leer una cadena:

```
char nombre[40];
scanf("%[^\n]", nombre);
printf("%s", nombre);
```

Entrada: Francisco Javier

Salida: Francisco Javier

Entrada con formato [scanf( )]

```
   scanf("%s %d %f", cadena, &n, &x);

Los datos se pueden introducir de las siguientes maneras.
```

Mensaje 120 0.25	Mensaje 120 0.25	Mensaje 120 0.25

Entrada y Salida sin formato

Funciones	DESCRIPCIÓN
var_char=getchar();	Lee un carácter de teclado, espera un salto de carro.
var_char=getche();	Lee un carácter con eco, no espera salto de carro.
var_char=getch();	Lee un carácter sin eco, no espera salto de carro.
gets(var_cadena);	Lee una cadena del teclado.
putchar(var_char);	Muestra un carácter en pantalla.
puts(variables);	Muestra una cadena en pantalla.

Estructuras de Control [if – then – else]

```
if (expresión)
  {
   proposiciones;
  }
proposición siguiente;
```

```
if (grado>=90)
  {
   printf(“\n FELICIDADES”);
  }
printf(“\n su grado es %d”, grado);
```

Estructuras de Control [if – then – else]

```
if (expresión)
  {
   proposición_1;
  }
else {
       proposición_2;
     }
proposición_siguiente;
```

```
if (x<=y)
   {
     min=x;
   }
else {
      min=y;
     }
printf(“min=%d\n”,min);
```

Estructuras de Control [if – then – else]

```
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);
            }
     }
```

Estructuras de Control [if – then – else]

```
(condición) ? (expresión1) : (expresión2) ;
```

```
if (x<=y)
   {
     min=x;
   }
else {
      min=y;
     }
printf(“min=%d\n”,min);
```

```
(x<=y) ? min=x : min=y;
printf(“min=%d\n”,min);
```

Estructuras de Control [switch]

Permite ejecutar una de varias acciones, en función del valor de una expresión. La sintaxis de esta proposición es:

```
switch (expresión-test)
   {
    case constante1: sentencia;
    case constante2: sentencia;
    case constante3: sentencia;
     . . .
    case constante_n: sentencia;
    default: sentencia;
   }
```

```
switch(ch = getchar())
{
 case 'a': printf("Azul");
            break;
 case 'b': printf("Blanco");
             break;
 case 'r': printf("Rojo");
             break;
 default : printf("Error");
}
```

Estructuras de Control [El ciclo – for -]

Es una estructura derepetición y se utiliza cuando se sabe exactamente el número de veces que se repetirán una serie de sentencias

```
for(inicialización; condición; incremento)
   {
     sentencia1;
     sentencia2;
     . . . . .
     sentencia_n;
   }
```

inicialización es una proposición de asignación para establecer el valor inicial de la variable de control.

Condición es una expresión relacional o lógica que determina cuando terminará el ciclo de repetición.

Incremento define cómo cambiará la variable de control o las variables de control, cada vez que cambia este.

Estructuras de Control [El ciclo – for -]

```
for(digito=0; digito<=9; ++digito)
   printf("%d \n", digito);
```

Programa que visualiza los dígitos del 0 al 9.
Ver ejemplo (while).

```
for( ; ; )  printf("Esto no termina");

for( ;cañas <=20; )
  { cañas += 1;
   borachera *=2;
  }

for(cont=0, total=0; cont<10; ++cont)
 {total += cont;
printf("cont=%d,total=%d\n",cont,total);
 }
```

1. bucle infinito.

2. Sin índice ni su actualización, pero se actualiza dentro del cuerpo del bucle.

3. Dos contadores en el índice.

Estructuras de Control [El ciclo – while -]

Esta proposición ejecuta una sentencia simple o compuesta, cero o más veces dependiendo del valor de una expresión. Su sintaxis es la siguiente:

```
while (expresión)
   {
    sentencia1;
    sentencia2;
    . . . . .
    sentencia_n;
   }
```

Una proposición while se ejecuta mientras la expresión sea verdadera (cualquier valor distinto de cero). En el momento en que la expresión sea falsa, entonces se ejecuta la siguiente línea después del fin del ciclo.

Estructuras de Control [El ciclo – while -]

```
int digito = 0;
while(digito <= 9){
   printf("%d \n", digito);
   ++digito;
}
```

Programa que visualiza los Dígitos del 0 al 9.

```
int digito = 0;
while(digito <= 9)
   printf("%d \n", digito++);
```

Una variante más concisa que la primera.

Estructuras de Control [El ciclo – do-while -]

Esta proposición ejecuta una sentencia o varias, una o más veces dependiendo de l valor de una expresión. Su sintaxis es:

```
do {
      sentencia1;
      sentencia2;
      . . . . .
      sentencia_n;
   } while (expresión);
```

En la sentencia *do-while* se ejecuta primero la sentencia o bloque de sentencias que estan dentro del *do*. Después se evalúa la *expresión* y si ésta es falsa termina la proposición *do-while*, sino se repote la sentencia o sentencias que están dentro del *do*.

Estructuras de Control [El ciclo – do-while -]

```
int n;
do {
     printf(“\n da un número: “);
     scanf(“%d”, &n);
   } while (n>100);
```

La sentencia “break”

La sentencia break: Esta sentencia ya descrita en el uso del switch, también sirve para terminar ciclos producidos por el for, while y do-while, antes de que se cumpla la condición normal de terminación. Por ejemplo:

```
for (t=0; t<100;t++)
   {
    contador 1;
    do{
       printf( “%d”, contador);
       contador++;
       if contador(==10)
          break;
      }while (1);
   }
```

La sentencia “continue”

Con esta sentencia, en vez de terminar um ciclo, termina con la iteración en curso saltando el resto de la pasada, es decir, todas las sentencias que se encuentran después de *continue* son ignoradas, para ir a la siguiente iteración. Igual que *break*, se puede utilizar en los ciclos for, while y do-while.

```
do{
        printf(“Da un numero: “);
        scanf(“%d”,&x);
        if (x<0) continue;
        printf(“El numero es %d”,x);
      }while (x!=100);
```

La sentencia “continue”

```
char c ;
for(;;){
   c = getch(); /*lee sin eco */
   if(c=='a' || c=='e' || c=='i'||
      c=='o' || c=='u')
     continue;
   putchar(c);
   putchar(c);
 }
```

En este programa, se leen los caracteres que se pulsen.
Si se pulsa una vocal minúscula, se realiza un salto al comienzo del bucle, en caso contrario el carácter se escribe dos veces en pantalla.

La función “exit( )”

La función exit(): Otra forma de terminar un ciclo de repetición desde dentro es utilizar la función exit(). A diferencia con el *break*, la función *exit()* terminará con la ejecución del programa y regresará el control al sistema operativo.

```
for(i=0; i<=1000; i++)
   {
    printf(“El valor de i es %d”, i);
    if (i>10) exit();
   }
```