Introducción al lenguaje C

1. Presentación del lenguaje C
2. Estructura de un programa en C
3. Tipos de datos escalares
4. Operadores básicos
5. Introducción a las funciones E/S
6. Condicionales
7. Iteraciones
8. Arrays (vectores o matrices)
9. Funciones y argumentos

1. Presentación del lenguaje C

- Creado en 1972 por D. Ritchie
- Lenguaje de propósito general
- Portátil o transportable (generalmente)
- Inicialmente de nivel medio (entre alto nivel y ensamblador)
- Pensado para (gestionar / programar) sistemas/comunicaciones
- Lenguaje compilado (compilar + enlazar)
- Modular (permite usar bibliotecas propias o estándar que se enlazan con nuestros programas principales)
- (Demasiado) conciso
- (Relativamente) sencillo de aprender

- **28 palabras reservadas:**

auto	**double**	**if**	**static**
break	**else**	**int**	**struct**
case	**entry**	**long**	**switch**
char	**extern**	**register**	**typedef**
continue	**float**	**return**	**union**
default	**for**	**sizeof**	**unsigned**
do	**goto**	**short**	**while**

2. Estructura de un programa

Algoritmo principal es
<declaración tipos>
<declaración variables>
Inicio
<composición secuencial acciones>
Fin

[int] main([<parámetros línea comando>])
[<declaración de parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>
}

- Un programa en C es una colección de una o más funciones (algoritmos con nombre; no hay subrutinas o procedimientos, esto es, algoritmos que no devuelven valores) que se llaman unas a otras, siendo siempre **main()** el nombre de la principal**.**

[<tipo>] nombre_función ([<lista parámetros>])
[<declaración de lista parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>
[return <expresión tipo de la función>]
}

- Al finalizar una función ésta devuelve el control a la función que la ha llamado o al S.O. en el caso de la función **main()**.

- Formato libre de líneas (no hay límite de tamaño)
- Las sentencias deben separarse mediante punto y coma **;**
- Las llaves {} agrupan conjuntos de sentencias lógicamente relacionadas (como inicio-fin, hacer-fin para,...)
- Los comentarios se ponen **/* así */**
- Todas las funciones llevan un tipo (**int,** por defecto)
- La instrucción **return** nos permite devolver **explícitamente** un valor. Si no se especifica, se devolverá un valor arbitrario, del tipo de la función.
- Es **obligatorio** definir **todas** las **variables** que se van a utilizar (equivale a **IMPLICIT NONE** de FORTRAN 90).

3. Tipos de datos escalares

- **Tipos básicos**
  - **char**, carácter
  - **int**, entero
  - **float**, real de simple precisión
  - **double**, real de doble precisión
  - NO HAY LÓGICOS

- **Tipos derivados de los básicos**
  - signed char, unsigned char --> como int
  - short, signed short, unsigned, unsigned short: short int, signed short int, unsigned short int, signed int, unsigned int
  - long, signed long, unsigned: long int, signed long int, unsigned long int

- **Declaración de variables o parámetros**

  <tipo> <variable> {, <variable>};

```
#include <stdio.h>
main() {
int entero1, entero2;
char caracter1, c, car2;
float real1, r2;
double d1, d2;

short int s;
long int entero_largo;
unsigned char cc;
return 0;
}
```

- **Declaración de variables o parámetros vectoriales (apartado 8)**

  <tipo> <variable> {[dim]}$^{+}$;

```
char cadena[20];
```

4. Operadores básicos

- **Asignación interna en C (=)** (:= en algorítmico, = en FORTRAN)

```
#include <stdio.h>
Int main() {
int entero = 1, otro_entero;
float r1, r2 = 1.2e-5;
double d=1.23456789;
char c = 'a', nueva_linea = '\n';

otro_entero = 12;
r1 = r2 / 0.5;
}
```

- Declaración de constantes

```
#define PI 3.1416
#define TAMANO_MAX 200
```

- **Aritméticos**

suma:	a + b	**a = a + 1**	**a += 1**	
resta:	c **-** d	**c = c - d**	**c -= d**	
incremento	**a=a+1**	**a++**		**++a**
decremento	**b=b-1**	**b--**		**--a**

Los operadores auto(incremento/decremento) ++a/a++ no son equivalentes

multiplicación:e ***** f	e = e * 2	e *= 2	
división:	g **/** h	g = g / 10	g /= 10
módulo/resto: i **%** 14	i = i % 2	i %= 2	

- **Relacionales/Comparación**

mayor:	j **>** k
mayor o igual: ll **>=** m	
menor:	n **<** op
menor o igual:	q **<=** r
igual:	**s == t (es doble signo =, no confundir con =)**
distinto:	**u != v**

- **Lógicos** (falso == 0, cierto es != 0)

  and: w **&&** y

  or: w **||** y

  negación: **!** z

- **Conversiones de tipos**
  - conversión al tipo de mayor precisión
  - los char se tratan como short int
  - ahormado o *cast*:
    (tipo_t) <expresión>
    fuerza a que el tipo resultado de evaluar **<expresión>** sea de **tipo_t**
    res = (int) real / entero;
    res2 = (float) entero / 2;
- En las bibliotecas (#include <math.h>,...) existen otras funciones para realizar operaciones: sqrt(), sin(), cos(),...
  - además existen funciones estándar para forzar conversiones

5. Introducción a las funciones E/S

Escritura:

int printf (“<cadena control>”, <argumentos>);

- La cadena de control especifica el formato y el número de argumentos
- Los argumentos son las variables o expresiones a escribir
- Devuelve nº de argumentos correctamente escritos
- En la cadena de control pueden aparecer:
  - constantes carácter o cadena, que aparecen como tales,
  - constantes tipo carácter: \b, \n, \t, \’, \”,...
  - descriptores de formato, **%?**, que indican el formato con el que mostrarán los argumentos (equivalente al format de FORTRAN), donde **?** es uno de los siguientes:

Código formato	Descripción
%c	carácter sencillo
%d	entero
%e	real en notación científica
%f	real simple precisión en notación científica
%g	el más corto de %e, %f
%o	octal
%x	hexadecimal
%s	cadena de caracteres
%u	decimal sin signo

Los descriptores se pueden especificar mediante
%m.n?
Ejemplos:
%10d%10.5f %20s

Ejemplos:

```
#include <stdio.h>
int main(){
  int entero, entero1, entero2;
  char carácter = 'c';

    entero = entero1 = entero2 = 1;
     printf ("Un entero en una linea: %d \n", entero);
     printf ("Dos enteros en una linea: %d, %d\n", entero1, entero2);
     printf ("Una cadena %s de caracteres.\n", "CADENA");
     printf ("Varios %d tipos %c mezclados %s\n", entero, caracter, "FRASE");
     }
```

Lectura:

int scanf (“<cadena de control>”, <argumentos>);

- cadena de control: idem que en el printf
- Devuelve número de argumentos leídos correctamente
- Los argumentos son las variables o expresiones a leer.
- **Los argumentos que sean de tipo dato-resultado o resultado y sean de tipos escalares, deben llevar delante el operador &**:
  &: indirección, pasa la dirección de la variable y no su valor.
  Esta es la forma en la que C modifica los valores de los argumentos de una función:

Ejemplo:

```
int  entero, entero1, entero2, ent3;
char c, cadena[20 ], string[40];

scanf (“%d”, &entero);    /* lee un entero */
scanf (“%d, %d”, &entero1, &entero2);        /* ha leido dos enteros*/
scanf (“%s”, cadena);      /* leida una cadena, que no es escalar*/
scanf (“%d %c %s\n”, &ent3, &c, string);   /* lee un entero, ent3, y un carácter, c,
        ambos escalares, y una cadena, string, que no es escalar*/
```

- Descriptores de formato para la lectura con scanf

Código formato	Descripción y argumento
%s	cadena de caracteres: string **(sin &)** porque es un array de caracteres
%c	carácter sencillo: &caracter
%d	entero: &entero
%e, %f, %g	real en notación científica, con signo y exponente opcionales: &real
%o	octal: &octal
%x	hexadecimal: &hexa
%u	decimal sin signo: &sin_signo

6. Condicionales

- **CONDICIONAL SIMPLE**

  if (<condición != 0>) <sentencia_1>

  if (<condición != 0>) { <composición_secuencial> }

- **CONDICIONAL COMPUESTO**

```
if (<condición != 0>) {
  <sentencias_cierto>
}
else {
  <sentencias_falso>
}
```

- **CONDICIONAL GENERALIZADO**

```
switch (<expresión_escalar>) {
  case <cte.1>: <sentencias>; [break;]
  case <cte.2>: <sentencias>; [break;]
  ...
  default: sentencias;
}
```

7. Iteraciones

- **for** (equivalente al **para** algorítmico)

```
para <var> desde <v_ini> hasta <expr. fin> hacer
        <sentencias para>
fin para
```

EQUIVALE A

```
for (<var = v_ini> ; <expr. fin> ; <expr. incr.>)
  <1_sentencia_for> |
for (<var>=<v_ini>; <expr. fin>;  <expr. incr.>) {
 <n_sentencias_for>
}
```

```
for (i=1; i <= 100; i++) {
  printf ("%d \t", i);
  if (i % 25 == 0) printf ("\n");
}
```

- **while** (equivale al **mientras** algorítmico)

mientras <condicion> sea cierta **hacer** <sentencias>
fin mientras

while (<condición != 0>) <sentencia_while>
|
while (<condición != 0>) {
 <sentencias_while>
}

```
i=1;
while (i != 100)
  printf ("\n%d", i++);

i=100;
while (i-- != 1)
  printf ("\n%d", i);
```

- **do while** (equivale al **repetir** algorítmico)

repetir
    <sentencias>
hasta que <condición>

do
    <sentencia_do_while>
while (<condición> != 0) |
do {
    <sentencias_do_while>
} while (<condición> != 0)

```
do {
scanf ("%d", &num);
} while (num);

do {
scanf ("%d", &num);
} while (num != 0);
```

8. Arrays

Podemos tener en C vectores de cualquier tipo básico o compuesto:

<tipo> <variable> {[<dimension>]}$^{+}$;

- Los índices variarán (0..N-1)
- C no comprueba que se exceda el número de elementos del vector
- Los nombres de los vectores (ej. cadena) son equivalente a la dirección en memoria donde comienza el vector (vector[0]).
- &enteros[i] accede a la dirección del elemento i en el vector enteros (→ véase paso por referencia).

```
int enteros[10], matriz[10][20];
char cadena[20];

matriz[9][19] = 12;
cadena[0] = 'c';
for (i=0; i<10; i++)  {
        scanf ("%d", &enteros[i]);
        printf("\n%d", enteros[i]);}
scanf ("%s", cadena);
```

9. Funciones y argumentos

- En C sólo hay funciones para representar algoritmos con nombre:

[<tipo>] nombre_función ([<lista parámetros>])
[<declaración de lista parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>
[return <expresión tipo de la función>]
}

- Todas las funciones devuelven un valor (que define su tipo). Por defecto, devuelve un entero (int) que no hace falta declarar
- Mediante **return** se devuelve (un valor concreto y) el control a la función que la haya llamado.

- Existen dos formas de pasar argumentos:
  - por valor (equivalente a par. dato), y es la forma por defecto → no modifica el valor del argumento
  - por referencia (equivalente a par. dato-resultado) → se puede modificar el valor del argumento.

- El paso por referencia implica pasar la dirección de la variable (direccionamiento indirecto) mediante el operador **& (indirección)**.

- Ejemplo, en el caso de *printf* y *scanf*, que son funciones de la biblioteca estándar de C, los argumentos se pasan:
  - por valor a printf, ya que no los modifica
  - por referencia (mediante &) a scanf porque sí los modifica

```
/* x e y son dos enteros que se pasan por valor a la función multiplica
   res es un entero, que se pasa por referencia, y se modifica con el
       resultado de la multiplicación
   x, y, resp son enteros que se pasan por valor a la función printf */
#include <stdio.h>
/* a y b son enteros y se pasan por valor → no se modifican
   c es un puntero a entero (dirección a un entero). Para modificarlo hay
       que acceder a su contenido mediante *c
   El tipo void de la función indica que no devolverá ningún valor */
 /* sintaxis equivalente multiplica(int a, int b, int* c)*/
void multiplica (a, b, c)
int a, b, *c;
{
  *c = a * b;
}

main() {
int x, y, res;
x = 10; y = 20;
multiplica (x, y, &res);
printf ("\nEl resultado de multiplicar %d por %d es %d", x, y, res);
}
```