

Programación I

Tipos de datos y operadores básicos

Iván Cantador

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Contenidos

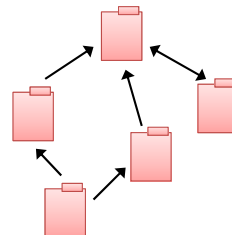
1

- Fundamentos de C
- Variables y tipos de datos
- Entrada/salida
- Operadores
- Macros

Fundamentos de C (I)

2

- La **programación modular** consiste en descomponer la complejidad de una aplicación informática en distintos componentes o *módulos*, donde cada módulo contiene un conjunto de funcionalidades relacionadas
 - Ejemplo. Un **programa de gestión de entradas de cine** podría tener módulos principales como:

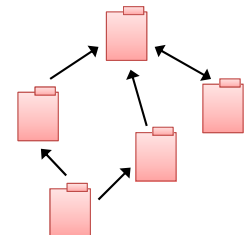


Fundamentos de C (I)

3

- La **programación modular** consiste en descomponer la complejidad de una aplicación informática en distintos componentes o *módulos*, donde cada módulo contiene un conjunto de funcionalidades relacionadas
 - Ejemplo. Un **programa de gestión de entradas de cine** podría tener módulos principales como:

- Interfaz gráfica de usuario
- Gestión de películas
- Gestión de salas
- Gestión de transacciones de pago
- Impresión de entradas



- En C cada módulo de un programa está compuesto (en general y al menos) por 2 ficheros:

- Fichero de cabecera (header)**

- De extensión **.h**
- Contiene la definición de tipos de datos y macros, y la declaración de funciones del módulo

- Fichero fuente (source)**

- De extensión **.c**
- Contiene la definición (implementación) de las funciones declaradas en el fichero de cabecera

```
/* complejo.h */

#ifndef COMPLEJO_H
#define COMPLEJO_H

/* Macros */

#define PI 3.1416

/* Definicion de tipos de datos estructurados */

typedef struct _Complejo {
    double re;
    double im;
} Complejo;

/* Declaracion de funciones */

double moduloComplejo(Complejo *a);

Complejo *sumaComplejos(Complejo *a, Complejo *b);

#endif
```

```
/* complejo.c */

#include <stdio.h>
#include <math.h>
#include "complejo.h"

/* Definicion de funciones */

double moduloComplejo(Complejo *c) {
    double modulo;

    if( !c ) return -1;

    modulo = sqrt(c->re * c->re + c->im * c->im);

    return modulo;
}

Complejo *sumaComplejos(Complejo *a, Complejo *b) {
    Complejo *s = NULL;

    if( !a || !b ) return NULL;

    s = (Complejo *) malloc(sizeof(Complejo));
    if( !s ) return NULL;

    s->re = a->re + b->re;
    s->im = a->im + b->im;

    return s;
}
```



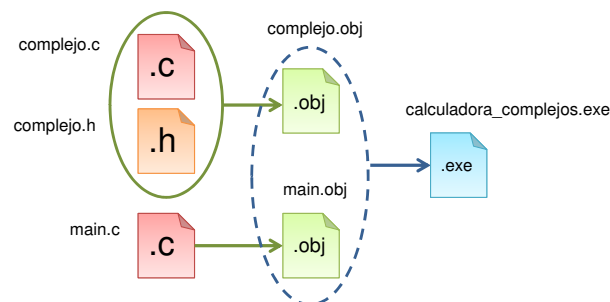
- La generación de un programa escrito en C se lleva a cabo en 2 pasos:

- Compilación**

- ficheros de **código fuente** (.c) → ficheros **objeto** (.obj)

- Enlazado o "lincado" (link)**

- ficheros **objeto** (.obj / .o) → fichero **ejecutable** (.exe)



- Directiva #include**

- Incluye el contenido de un fichero en el punto del programa donde se encuentra el comando
- El fichero a incluir será en general de cabecera de una "librería" de funciones

- Sintaxis**

```
#include <nombre_fichero> /* Librerías de sistema */
#include "nombre_fichero" /* Librerías de usuario */
```

- Librerías de sistema habituales**

- **stdio.h** Entrada/Salida (E/S, I/O) estándar
- **stdlib.h** Funciones y tipos de datos generales
- **math.h** Funciones matemáticas
- **string.h** Funciones para manejo de cadenas de caracteres
- **time.h** Funciones y tipos para fechas, horas, tiempo del sistema
- **mem.h** Funciones para manejo de memoria

- Todo programa en C ha de tener una función llamada **main**, que es la da comienzo a la ejecución del programa

```
#include <stdio.h>

void main() {

    printf("Hola mundo!\n");    // Escribe en pantalla
                                // Hola mundo!


}
```

- Todo programa en C ha de tener una función llamada **main**, que es la da comienzo a la ejecución del programa
 - Dato de **salida**
 - Tipo: *void, int*
 - Valor devuelto al sistema operativo por el programa al acabar su ejecución
 - Argumentos de **entrada** (se pueden omitir)
 - **argc**: un entero con número de argumentos del programa + 1
 - **argv**: una lista de cadenas de caracteres con:
 - el nombre (argv[0]) del programa
 - los argumentos de entrada (argv[1], argv[2], ...) del programa

Fundamentos de C (VII)

10

- Función **main** con argumentos de entrada



```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i;

    printf("Nombre del programa: %s\n", argv[0]);
    printf("Numero de argumentos: %d\n", argc-1);
    printf("Argumentos:\n");
    for ( i=1; i<argc; i++ ) {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```

- Ejemplo de ejecución del programa en línea de comandos

```
C:\>mi_ejecutable.exe A b C
Nombre del programa: mi_ejecutable.exe
Numero de argumentos: 3
Argumentos:
A
b
C
```



Contenidos

11

- Fundamentos de C
- **Variables y tipos de datos**
- Entrada/salida
- Operadores
- Macros

Declaración de variables (I)

12

- Para declarar una variable se especifica su tipo y nombre

```
short x;           // numero entero corto
int y;             // numero entero
long z;            // numero entero largo
float r1;          // numero real
double r2;         // numero real largo
char caracter;     // caracter
char cadena1[256]; // cadena con memoria para 256 caracteres
char *cadena2;     // cadena sin memoria todavía
```


- En una línea se pueden declarar varias variables del mismo tipo

```
int entero1, entero2;
```

Declaración de variables (II)

13

- Las variables se declaran al principio de un bloque de instrucciones → ¡¡antes de las instrucciones!!




```
/* El siguiente bloque de instrucciones es correcto */
double x;                               // declaracion de x
x = 2;                                  // uso de x
printf("El numero es: %d", x);
```

Declaración de variables (II)

14

- Las variables se declaran al principio de un bloque de instrucciones → ¡¡antes de las instrucciones!!



```
/* El siguiente bloque de instrucciones es correcto */
double x;                               // declaracion de x
x = 2;                                  // uso de x
printf("El numero es: %d", x);
```

```
/* El siguiente bloque de instrucciones es incorrecto
porque la variable x se declara despues de su uso;
en este caso, la asignación de un valor */
x = 2;                                  // uso de x
double x;                               // declaracion de x
printf("El numero es: %d", x);
```

Nombrado de variables

15

- El nombre de una variable:
 - Sólo puede tener caracteres alfanuméricos ('a', 'A', '1', '2', ...) y el carácter '_'
- ```
int edad;
int edad1;
int edadCliente;
int edad_cliente;
```
- No puede empezar por carácter numérico
- No permite:
  - Letras acentuadas: 'á', 'Á', 'à', 'ä', ...
  - Letras como la 'ñ' y la 'Ñ'
  - Otros símbolos: '(', '[', '#', '@', '&', '+', '-', '/', '\*', '?', ...

- En la declaración también se puede inicializar (asignar el valor inicial de) una variable

```
int numeroEntero1 = -3, numeroEntero2;
double numeroRealLargo = 6.5;
char caracter = 'a';
char *cadenaCaracteres = "Hola";
```

- El tamaño en Bytes de una variable o de un tipo de dato se puede obtener mediante el operador **sizeof** (que no es válido para cadenas de caracteres, char \*)

```
int numeroEntero = -3;
double numeroReal = 6.5;
char caracter = 'a';
char *cadena = "Hola";

printf("Tamano de numeroEntero = %d\n", sizeof(numeroEntero));
printf("Tamano de un int = %d\n", sizeof(int));

printf("Tamano de caracter = %d\n", sizeof(caracter));
printf("Tamano de un char = %d\n", sizeof(char));

printf("Tamano de numeroReal = %d\n", sizeof(numeroReal));
printf("Tamano de un double = %d\n", sizeof(double));

// Para obtener la longitud de una cadena de caracteres se
// usa la funcion strlen, declarada en string.h
printf("Tamano de cadena = %d\n", strlen(cadena));
```

## Tamaño de tipos de datos

18

| Tipo de dato | Nombre | Bytes* | Descripción                                                                         | Ejemplos                                                                                      |
|--------------|--------|--------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Carácter     | char   | 1      | Representa un carácter ASCII, aunque se puede usar como un número entre -128 y +127 | char c1, c2;<br>c1 = 'a';<br>c2 = 'A' + 5;<br>printf("%c", c2);<br>printf("%d", c1);          |
| Entero corto | short  | 2      | Entero de 16 bits con signo                                                         | short s1 = 0;<br>printf("%d", s1);                                                            |
| Entero       | int    | 4      | Entero de 32 bits con signo                                                         | int i1 = 0;<br>i1 = 288;<br>i1 = 032; /*Octal*/<br>i1 = 0xf0; /*Hexad.*/<br>printf("%d", i1); |
| Real         | float  | 4      | Número real en coma flotante                                                        | float f1 = 0.0;<br>f1 = 3.142;<br>f1 = 3e-12;<br>printf("%f", f1);<br>printf("%5f", f1);      |
| Real doble   | double | 8      | Número real en coma flotante, con doble precisión                                   | double d1 = 0.0;<br>d1 = 4.3e-3;<br>printf("%f", d1);<br>printf("%4.3f", d1);                 |

\* Puede variar dependiendo del microprocesador y sistema operativo. Ver operador **sizeof**.

## Conversión entre tipos de datos

19

- Conversión de tipos de datos
  - Implícita: la expresión se convierte al tipo de mayor rango

double ← float ← long ← int ← short ← char

```
int i = 9 / 2; // El resultado es 4
int i = 9.0 / 2; // Error de compilacion!!
float f = 9.0 / 2; // El resultado es 4.5
```

## • Conversión de tipos de datos

- **Implícita:** la expresión se convierte al tipo de mayor rango

double ← float ← long ← int ← short ← char

```
int i = 9 / 2; // El resultado es 4
int i = 9.0 / 2; // Error de compilacion!!
float f = 9.0 / 2; // El resultado es 4.5
```

- **Explícita:** la expresión se convierte mediante un “*casting*”  
(tipo) expresion

```
int i = (int) (9.0 / 2); // El resultado es 4
float f = (float) 9 / 2; // El resultado es 4.5
```

## • Conversión de tipos de datos

- **Implícita:** la expresión se convierte al tipo de mayor rango

double ← float ← long ← int ← short ← char

```
int i = 9 / 2; // El resultado es 4
int i = 9.0 / 2; // Error de compilacion!!
float f = 9.0 / 2; // El resultado es 4.5
```

- **Explícita:** la expresión se convierte mediante un “*casting*”  
(tipo) expresion

```
int i = (int) (9.0 / 2); // El resultado es 4
float f = (float) 9 / 2; // El resultado es 4.5
```

→ El *casting* puede conllevar a la pérdida de información

```
int i = 2000000;
short s = (short) i; // El valor de i excede el límite de s
```

# Contenidos

- Fundamentos de C
- Variables y tipos de datos
- **Entrada/salida**
- Operadores
- Macros

# Entrada/salida (I)

- **Escritura por pantalla mediante la función `printf`, declarada en `stdio.h`**

```
char caracter = 'a';
int entero = 0;
float real1 = 1.0;
double real2 = 2.0;
char *cadena = "Hola";
```

```
printf("%c", caracter);
printf("%d", entero);
printf("%f", real1);
printf("%f", real2);
printf("%s", cadena);
printf("Un entero: %d y un real: %f", entero, real1);
printf("Dos caracteres %c %c con fin de línea.\n", caracter, caracter);
```

## • Escritura en pantalla: printf

```
printf(<cadena>);
printf(<cadena_con_formato>, <expr1>, <expr2>, ...);

printf("Hola!\n");
printf("Hola %s!\n\tHoy es %d de %s\n", nombre, dia, mes);
```

## • Escritura en pantalla: printf

```
printf(<cadena>);
printf(<cadena_con_formato>, <expr1>, <expr2>, ...);

printf("Hola!\n");
printf("Hola %s!\n\tHoy es %d de %s\n", nombre, dia, mes);
```

## • Ejemplos de formateo

```
printf("Un numero entero: %d\n", entero);
printf("Un numero entero: %d\n", (entero + 10) / 2);
printf("Un numero entero sin signo: %u\n", entero);
printf("Un numero entero en hexadecimal: %x\n", entero);
printf("Un numero real: %f\n", real);
printf("Un numero real en forma exponencial: %e\n", real);
printf("Un numero real ajustando decimales: %g\n", real);
printf("Un carácter: %c\n", caracter);
printf("Un cadena de caracteres (string): %s\n", cadena);
```

## • Lectura desde teclado: scanf

```
scanf(<cadena_con_formato>, &<var1>, &<var2>, ...);

int entero1, entero2, numLeidos;
float real;
char cadena[32];

numLeidos = scanf("%d", &entero1);
numLeidos = scanf("%d %f", &entero2, &real);
numLeidos = scanf("%s", cadena); /* OJO: NO &cadena */
/* en scanf no se pone '&' */
/* con cadena de caracteres */
```



## • Ejemplo de lectura por teclado y escritura por pantalla de un dato

### • En Scratch



### • En C

```
#include <stdio.h>

void main() {
 int x;
 printf("Introduce un número: ");
 scanf("%d", &x);
 printf("El número introducido es %d", x);
}
```

- Fundamentos de C
- Variables y tipos de datos
- Entrada/salida
- Operadores**
- Macros

| Tipo de operador            | Operadores                                                                                                        | Descripción                                                                                                               | Ejemplos                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| <i>Aritméticos</i>          | Suma: +<br>Resta: -<br>Multiplicación: *<br>División: /<br>Módulo: %                                              | Operadores aritméticos elementales<br><br>Más operaciones en las librerías matemáticas                                    | <pre>char c = 'A'-'a'; int i = c%4; double d = (2/3)*3;</pre>                |
| <i>Relacionales</i>         | Menor: <<br>Menos o igual: <=<br>Mayor: ><br>Mayor o igual: >=<br>Igual: ==<br>Distinto: !=                       | Operadores de comparación entre enteros, caracteres y reales                                                              | <pre>int cond1 = (32&gt;=4); int cond2 = (cond1&lt;6); cond1 == cond2;</pre> |
| <i>Booleanos</i>            | NOT lógico: !<br>AND lógico: &&<br>OR lógico:                                                                     | Operaciones lógicas, aplicadas sobre enteros, siendo 0 = <i>false</i> y otro valor <i>true</i>                            | <pre>int a, b, c; (!(a &amp;&amp; (b    c)))</pre>                           |
| <i>Operaciones con bits</i> | NOT nivel bit: ~<br>AND nivel bit: &<br>OR nivel bit:  <br>XOR nivel bit: ^<br>Despl. izq.: <<<br>Despl. der.: >> | Operaciones lógicas aplicadas a nivel de bit<br><br>En los desplazamientos: x << y, significa desplazar y bits el valor x | <pre>int a = 0xff, b = 0xf0; (a &amp; b); (a   b); a &lt;&lt; 2;</pre>       |

| Tipo de operador                    | Operadores                                                                                      | Descripción                                                                                                                      | Ejemplos                                                                       |
|-------------------------------------|-------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <i>Asignación</i>                   | Suma con asig.: +=<br>Resta con asig.: -=<br>Multiplic. con asig.: *=<br>División con asig.: /= | Operaciones de asignación a una variable<br><br>Las de forma x+=y son equivalentes a x=x+y                                       | <pre>int a, b, c; a = b = c = 0; a += 1; b += c += a;</pre>                    |
| <i>Incrementales</i>                | Preincremento: ++c<br>Postincremento: c++<br>Predecremento: --c<br>Postdecremento: c--          | Postincremento: devuelve el valor actual de c y luego hace c=c+1<br><br>Preincremento: hace c=c+1 y devuelve el nuevo valor de c | <pre>char c, d= 0; c = d++; d = c++;</pre>                                     |
| <i>Concatenación de expresiones</i> | expr1 , expr2                                                                                   | Sirve para unir varias expresiones en una misma línea de código                                                                  | <pre>int i; i = (i=2, i++, i*= 2);</pre>                                       |
| <i>Condicional</i>                  | exp1 ? exp2 : exp3                                                                              | Expresión condicional<br><br>Si exp1 es <i>true</i> el resultado es exp2, si no exp3                                             | <pre>int a, b, max, min; max = (a&gt;b ? a : b); min = (a&lt;b ? a : b);</pre> |

- Precedencia de operadores (de mayor a menor)**
  - Aritméticos
  - Relacionales
  - Booleanos
  - Asignaciones
- Precedencia de operadores aritméticos (de mayor a menor)**
  - () paréntesis
  - ^ exponenciación
  - \*/ multiplicación, división
  - + - suma, resta

```

a = 1 + 2 * 3; // El resultado es 7
a = (1 + 2) * 3; // El resultado es 9
b = 1 + 2 * 3 ^ 2 // El resultado es 19
b = 1 + (2 * 3) ^ 2 // El resultado es 37

```



- Fundamentos de C
- Variables y tipos de datos
- Entrada/salida
- Operadores
- **Macros**

- Directiva **#define**
  - Define una constante (**macro**) con o sin parámetros, con el fin de que se sus apariciones se sustituyan textualmente por su valor
  - Ejemplos de macros sin parámetros

```
#define PI 3.1416
#define ERR 0
#define OK 1
#define MENSAJE "Mensaje predefinido"
```
  - Ejemplos de macros con parámetros

```
#define CUADRADO(N) (N)*(N)
#define MAX(A,B) ((A)>(B)?(A):(B))

double p;
int i=1, j, k;
p = PI;
j = CUADRADO(i+1);
k = MAX(i*PI, j);
```