Tema 1. Fundamentos de C
Parte I

Diseño de Algoritmos

2º I.T.I. Electricidad
E.U. Politécnica

M. Carmen Aranda Garrido
Despacho: I-307

Departamento Lenguajes y Ciencias de la Computación.
Universidad de Málaga

Índice

1. Estructura de un programa.

PROGRAMA = DATOS + INSTRUCCIONES


```
órdenes al preprocesador (órdenes que comienzan con #)
declaraciones globales (tipos de datos y variables)
prototipos de funciones
main( ) {
   variables locales
   bloque
}
funcion1( ) {
   variables locales
   bloque
}
```


1.1. Comentarios.

Para poner comentarios en un programa escrito en **C** usamos los símbolos **/*** y ***/**:

```
/* Este es un ejemplo de comentario */

/* Un comentario también puede
estar escrito en varias líneas */
```

El símbolo **/*** se coloca al principio del comentario y el símbolo ***/** al final.

También se usa el símbolo // que pone un comentario desde ese símbolo hasta el final de la línea.

```
// Esto es un comentario
```

1.2. Palabras reservadas.

char	**int**	**float**	**double**
else	**do**	**while**	**for**
switch	**short**	**long**	**extern**
static	**default**	**continue**	**break**
register	**sizeof**	**typedef**	**if**

1.3. Identificadores.

Un identificador es el nombre que damos a las variables y funciones. Está formado por una secuencia de letras y dígitos, aunque también acepta el carácter de subrayado **_**. Por contra no acepta los acentos ni la **ñ/Ñ**.

Válidos	**No válidos**
_num	1num
var1	número2
fecha_nac	año_nac

2. Datos y Tipos de datos.

Datos = Información que maneja el programa.
Todos los DATOS deben tener asociado un TIPO DE DATO.

Tipos de datos SIMPLES o FUNDAMENTALES en C:

TIPO	Tamaño	Rango de valores
char	1 byte	-128 a 127
int	2 ó 4 bytes	-32768 a 32767 ó -2^{31} a $2^{31}-1$
float	4 bytes	3'4 E-38 a 3'4 E+38
double	8 bytes	1'7 E-308 a 1'7 E+308
void	Tipo nulo	
puntero	Contiene una dirección de memoria	

2. Datos y Tipos de datos (II).

- Calificadores de tipo:
  - signed (lleva signo)
  - unsigned (no lleva signo)
  - short (formato corto)
  - long (formato largo)

char, unsigned char, signed char
int, unsigned int, signed int, unsigned int, long unsigned int, short unsigned int
float
double,long double

3. Declaración de variables.

[calificador] <tipo> <nombre1>,<nombre2>=<valor>,...;

```
#include <stdio.h>

main() /* Suma dos valores */
{
        int num1=4,num2,num3=6;
        printf("El valor de num1 es %d",num1);
        printf("\nEl valor de num3 es %d",num3);
        num2=num1+num3;
        printf("\nnum1 + num3 = %d",num2);
}
```


3. Declaración de variables (II).

Variables de tipo puntero:

```
<tipo> *<nombre>;
```

Un puntero contiene la dirección de memoria donde hay un dato de un tipo concreto (tipo de dato al que apunta el puntero).

```
int *ptr1, *ptr2;
float *puntero;
```

3. Declaración de constantes

Las constantes tienen un identificador y un valor ÚNICO en todo el programa

```
#include <stdio.h>
#define pi 3.1416
#define escribe printf
main() /* Calcula el perímetro */
{
        int r;
        escribe("Introduce el radio: ");
        scanf("%d",&r);
        escribe("El perímetro es: %f",2*pi*r);
}
```


4. Operadores.

(1) Asignación: =, +=,-=,*=, /=, %=

(2) Aritméticos:

Suma	+ (int o float)
Resta	- (int o float)
Producto	* (int o float)
División	/ (int o float)
Resto de una división entera	% (int)
Incremento y decremento	++, -- (int)

(3) Relacionales:

Menor	<
Menor o igual	<=
Mayor o igual	>=
Mayor	>
Igual	==
Distinto	!=

Devuelven 0 ó 1

4. Operadores (II).

(4) Lógicos:

Negación	!A
Conjunción (AND)	A && B
Disyunción (OR)	A || B

Devuelven 0 ó 1

En C no existe tipo de datos lógico:

FALSO → 0
VERDAD → ≠0

(5) sizeof(): Devuelve el número de bytes que ocupa el tipo de dato o la variable que hay entre paréntesis.

(5) De punteros:

Operador de dirección	&
Operador de indirección	*

4. Operadores. Conversión de tipos (III)

- **Conversión IMPLÍCITA:** Cuando en una expresión se mezclan constantes y variables de distintos tipos el compilador convierte de forma **automática** todo a un único tipo siguiendo las siguientes reglas:
  - **Promoción**: en cualquier operación en la que aparezcan dos tipos diferentes se eleva el rango del que lo tiene menor para igualarlo al del mayor.
  - El rango o categoría de los tipos de mayor a menor es el siguiente:
    - double, float, long, int, short, char
    - Los tipos *unsigned* tienen el mismo rango que los tipos a los que están referidos
  - En una sentencia de asignación, el resultado final de los cálculos se reconvierte al tipo de la variable al que está siendo asignado. El proceso puede ser una promoción o una pérdida de rango según la categoría de la variable a la que se le efectúa la asignación.

4. Operadores. Conversión de tipos (IV)

- **Conversión EXPLÍCITA: Casting**

(tipo) expresión;

- Es una conversión **explícita** del tipo de una expresión realizada por el programador
- La forma general de realizarla es:

```
int x;
float y = 1.0;
float z = 2.0;
x = (int) y + (int) z
```


5. Funciones de Entrada/Salida

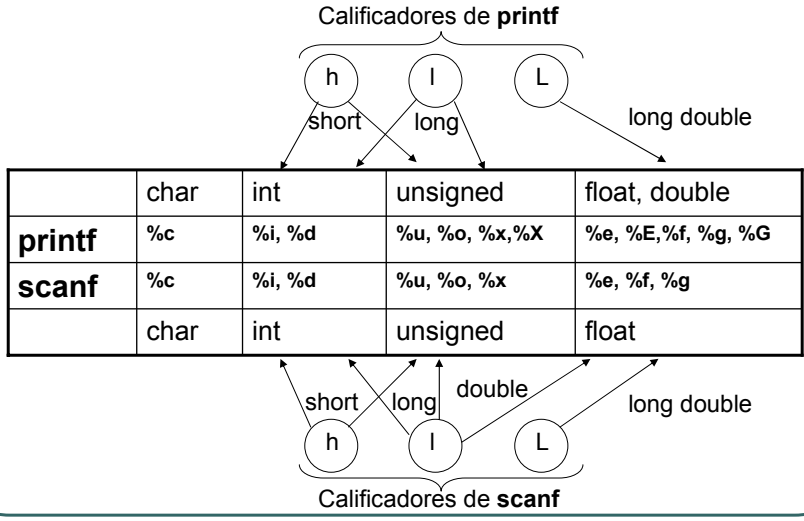
- Se encuentran en stdio.h.
- Función de ENTRADA o LECTURA con formato:

int scanf (const char *formato, &variable,...);

- Función de SALIDA o ESCRITURA con formato:

int printf (const char *formato, argumento,...);

5. Funciones de Entrada/Salida (II)



	char	int	unsigned	float, double
printf	**%c**	**%i, %d**	**%u, %o, %x,%X**	**%e, %E,%f, %g, %G**
scanf	**%c**	**%i, %d**	**%u, %o, %x**	**%e, %f, %g**
	char	int	unsigned	float

5. Funciones de Entrada/Salida (III)

- Lee un solo carácter de la entrada estándar (*stdin)*
- Devuelve un valor que es el carácter leído y que puede ser asignado a una variable de tipo *char* o *int*
- Guarda la entrada hasta que se pulsa *ENTER*
- Se encuentra en ***stdio.h***

int getchar (void);

- Lee un solo carácter del teclado
- No hace eco en la pantalla
- Se encuentra en ***conio.h***

int getch (void);

- Lee un solo carácter del teclado
- Hace eco en la pantalla
- Se encuentra en ***conio.h***

int getche (void);

- Escribe un carácter por pantalla
- Se encuentra en ***stdio.h***

int putchar (char c);

5. Funciones de Entrada/Salida (IV)

```
int main () {
   int c;
   while ((c=getchar() != '\n') {
        putchar(c);
   }
}
```

```
int main () {
   char c;
   c=getche();
   printf("Ha tecleado el carácter '%c'",c);
}
```

```
int main () {
   int edad;
   float sueldo;
   printf("Escriba su edad y sueldo.\n");
   scanf("%d %f", &edad, &sueldo);
   printf("\nEdad: %d Sueldo: %.2f \n", edad, sueldo);
}
```


5. Operaciones de Entrada/Salida cin-cout
#include <iostream> + using namespace std;

Entrada sin formato *(es el mismo tipo de la variable el que determina la E/S):*

***cin >> variable**;*

***cin >> variable1 >> variable 2 >> .....**;*

Salida sin formato *(es el mismo tipo de la variable el que determina la E/S):*

cout << variable; ***cout <<.....<<.....<<.....;***

***cout << mensaje**;*

5. Operaciones de Entrada/Salida cin-cout
#include <iostream> + using namespace std;

```
#include <iostream>
#include <stdio.h>
int main () {
   int edad;
   float sueldo;
   printf("Escriba su edad y sueldo.\n");
   scanf("%d %f", &edad, &sueldo);
   printf("\nEdad: %d Sueldo: %.2f \n", edad, sueldo);
   return 0;
}
```

Posible ejecución

```
Escriba su edad y sueldo.
33 1000
Edad: 33 Sueldo: 1000.00
Presione una tecla para continuar....
```

```
#include <iostream>
using namespace std;

int main () {
   int edad;
   float sueldo;
   cout << "Escriba su edad y sueldo.\n";
   cin >> edad >> sueldo;
   cout <<"\nEdad:"<<edad<<" Sueldo: "<<sueldo<<"\n";
   return 0;
}
```

Posible ejecución

```
Escriba su edad y sueldo.
33 1000
Edad: 33 Sueldo: 1000
Presione una tecla para continuar....
```


6. El Preprocesador de C

- Procesa el programa fuente antes de que éste sea tratado por el compilador
- Acciones posibles del preprocesador:
  - **Inclusión** de otros archivos en el archivo que se va a compilar
  - Definición de **constantes simbólicas**
  - Definición de macros
  - Compilación condicional de código de programa
  - Ejecución condicional de directivas de preprocesador
- Todas las órdenes dirigidas al preprocesador se llaman **directivas** y comienzan con el símbolo **#**. Como excepción, **en C++**, tenemos la "directiva" **using namespace std**.
- Cada directiva debe estar en una línea propia y antes de cada directiva sólo pueden aparecer espacios en blanco

6. El Preprocesador de C (II)

- Directiva #define

#define *identificador texto_a_reemplazar*

  - *Identificador*
    - Constante que queremos definir (macro)
  - *Texto_a_reemplazar*
    - Secuencia de caracteres que va a ser representada por la macro
    - Puede incluir expresiones con operadores
  - Todas las apariciones de *identificador* en el código fuente del programa son automáticamente sustituidas por *texto_a_reemplazar* antes de la compilación del programa

6. El Preprocesador de C (III)

```
#define MAXIMO 100
#define MINIMO 10
#define PI 3.141592
main() {
  char cadena[MAXIMO];
  char cad[MINIMO];
  ..............
```

```
#define LOG5 0.698970004
#define LN5 1.609437912
#define DOBLELN5 LN5*2
#define RESUL "\n- El resultado es el siguiente "
#define FIN printf("\n FIN.")
...
printf(RESUL);
printf("\n Logaritmo decimal de 5: %f", LOG5);
printf("\n Su mitad: %f", LOG5/2);
printf("\n Su cuadrado: %f", LOG5*LOG5);
printf("\n Doble del neperiano de 5: %f",
       DOBLELN5);
FIN;
```

6. El Preprocesador de C (IV)

- ***#define*** *permite definir órdenes, usualmente simples, que simplifican la construcción de programas y los hacen más rápidos que usando funciones. Pueden tener argumentos que se ponen entre paréntesis.*
- *Para evitar errores cuando la macro se usa en expresiones más complejas, se debe poner entre paréntesis cada argumento y la expresión completa.*
- *Este tipo de macroinstrucciones también puede implementarse usando funciones, pero las macros son independientes de tipo de dato.*
- *Las macros generan programas más grandes pero más rápidos.*

```
#define cuadrado(x) x*x
#define cubo(x) cuadrado(x)*x
#define ABS(X) ( (x>=0) ? x:0-x)
#define SUM_CUAD(X,Y) X*X + Y*Y
ERROR al evaluar SUM_CUAD(a-b,2)
Solución: #define SUM_CUAD(X,Y) (X)*(X)+(Y)*(Y)
ERROR al evaluar z/SUM_CUAD(2,3)
Solución: #define SUM_CUAD(x,y) ((x)*(x)+ (y)*(y))
```


6. El Preprocesador de C (V)

- Inclusión de ficheros

#include <*fichero*>

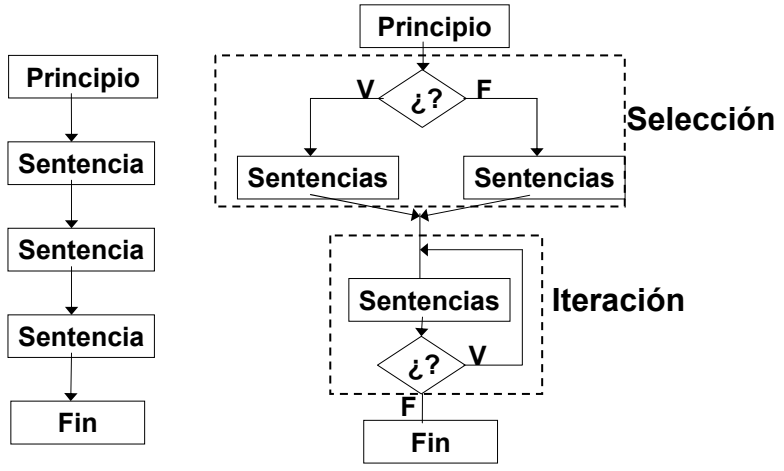
#include "*fichero*"

- En el lugar de la directiva #include se incluye una copia del archivo especificado antes de la compilación del programa
- "*fichero*"
  - Si el nombre del archivo va entre comillas el preprocesador lo buscará en el directorio donde se encuentra el archivo que se está compilando
  - Esta forma se utiliza para incluir archivos definidos por el programador
- *<fichero>*
  - Si el nombre del archivo va entre corchetes angulares el preprocesador lo buscará en algún directorio estándar del sistema
  - Esta forma se utiliza para incluir ficheros de cabecera de bibliotecas estándar (stdio.h, math.h,...). En C++ no hay que poner la extensión al incluir un fichero de cabecera (iostream,...)

7. Funciones matemáticas (math.h)

- **int abs (int);** • Devuelve el **valor absoluto** de un entero
- **double cos (double);** • Devuelve el **coseno**
- **double sin (double);** • Devuelve el **seno**
- **double exp (double x);** • Devuelve la exponencial: e^x
- **double fabs (double);** • Devuelve el **valor absoluto** de un real
- **double pow (double x, double y);** • Devuelve la potencia de x elevado a y: x^y
- **double sqrt (double x);** • Devuelve la **raiz cuadrada** positiva de x. Si x es negativo devuelve 0
- **double log (double x);** • Devuelve el **logaritmo neperiano**
- **double log10 (double x);** • Devuelve el **logaritmo en base 10**

8. Estructuras de control



Estructura secuencial

Estructuras no secuenciales

8. Sentencias de control selectivas (II)

```
if (condición) {
      Bloque Verdad
}
```

```
if (condición) {
      Bloque Verdad
}
else {
      Bloque Falso
}
```

```
if (condición1) {
      Bloque1
}
else if (condición2){
      Bloque2
}
else if (condición3){
      Bloque3
}
else {
      Bloque4
}
```


8. Sentencias de control selectivas (III)

```
switch (<expresión>) {
  case <valor1>: Bloque 1

break;
  case <valor2>: Bloque 2
            break;
  case <valor3>: Bloque 3
            break;
  default: Bloque 4;
}
```

Cada case puede incluir una o más sentencias sin necesidad de ir entre llaves, ya que se ejecutan todas hasta que se encuentra la sentencia **BREAK**. La variable evaluada sólo puede ser de tipo **entero** o **caracter**. **default** ejecutará las sentencias que incluya, en caso de que la opción escogida no exista.

8. Sentencias de control repetitivas (IV)

```
while (<condicion>) {
     Bloque de sentencias
}
```

```
do {
  Bloque de sentencias
}  while (<condicion>)
```

```
for(<inicialización>; <condición>; <operación>)
{
   Bloque de Sentencias;
}
```


8. Sentencia *break* (V)

- La sentencia **break** se usa para salir de un bucle.
- Se puede usar con cualquiera de los tres bucles.
- Cuando se utiliza un bucle infinito (`while(1)`, `do..while(TRUE)`, `for(;;)`), sólo se puede salir de ellos usando la sentencia `break`.
- Ejemplo:

```
/* Escribe el cubo de una lista de números hasta que lee un 0 */
#include <stdio.h>
void main() {
  int i;
  for(;;) {
    puts("Dame un número para calcular el cubo");
    scanf("%d", &i);
    if(i==0) break; /* para el bucle infinito */
    printf("El cubo de %d es %d\n", i, i*i*i);
  }
  puts("Has dado un cero");
}
```

8. Sentencia *continue* (VI)

- La sentencia **`continue`** se usa SÓLO dentro de un bucle.
- Termina la iteración que se esté ejecutando y se vuelve a evaluar la condición del `while`, del `do-while`, excepto en el `for` que va a la actualización.
- Ejemplo: Muestra los valores desde 0 hasta 9 excepto el 5.

```
for(i=0;i<10;i++) {
    if(i==5) continue;
    printf("El valor de i es %d\n", i);
 }
```


9. Declaración de funciones.

Su sintaxis es:

tipo_función nombre_función (tipo y nombre de argumentos)
{
bloque de sentencias
}

tipo_función: puede ser de cualquier tipo de los que conocemos.
El valor devuelto por la función será de este tipo.
Por defecto, es decir, si no indicamos el tipo, la función devolverá un valor de tipo entero (**int**).
Si no queremos que retorne ningún valor deberemos indicar el tipo vacío (**void**).

9.Tiempo de vida de los datos.

Según el lugar donde son declaradas puede haber dos tipos de variables.

Globales: las variables permanecen activas durante todo el programa. Se crean al iniciarse éste y se destruyen de la memoria al finalizar. Pueden ser utilizadas en cualquier función.

Locales: las variables son creadas cuando el programa llega a la función en la que están definidas. Al finalizar la función desaparecen de la memoria.

Si dos variables, una global y una local, tienen el mismo nombre, la local prevalecerá sobre la global dentro de la función en que ha sido declarada.

9. Declaración de funciones. Ejemplo

```
#include <stdio.h>

void funcion(void); /* prototipo */
int num=5; /* variable global */
main()  /* Escribe dos números */
{
       int num=10; /* variable local */
       printf("%d\n",num);
       funcion(); /* llamada */
}
void funcion(void)
{
       printf("%d\n",num);
}
```

9. Paso de Argumentos POR VALOR

- Cuando se produce la **Llamada a una Función**, se **transfiere la ejecución** del programa a la definición de la función. **Pasos:**
  - **1. Se declaran las variables de los argumentos formales.**
  - **2. Se COPIA el VALOR de los argumentos actuales en las variables de los argumentos formales.**
    - **Esta COPIA se hace por orden:** El primer argumento actual en el primer argumento formal, el segundo en el segundo...
    - **Esta COPIA NO SE HACE POR EL NOMBRE** de los respectivos argumentos formales y actuales.
    - Observe que se produce una **COPIA del valor:** Si el argumento actual es una variable, se copia su valor en el correspondiente argumento formal, pero ambos **argumentos actuales y formales son variables DISTINTAS.**
  - **3. Se declaran las variables locales a la función.**
  - **4. Se ejecuta el código de la función.**
  - **5. Al terminar la función las variables LOCALES son destruidas.**

9. Paso de Argumentos por Valor

```
#include <stdio.h>

int suma(int,int); /* prototipo */
main() /* Realiza una suma */{
  int a=10,b=25,t;
  t=suma(a,b); /* llama a la función y guarda el valor */
  printf("%d=%d",suma(a,b),t);
  suma(a,b); /* el valor se pierde */
}

/*************************************
 Función suma: suma dos números enteros
 Entrada: dos enteros
 Salida: un entero
*************************************/
int suma(int a,int b){
     return (a+b);
}
```

9. Paso de Argumentos por Referencia

- **RECUERDA:** En un **paso de argumentos por VALOR**, si los arg. formales se modifican, los arg. actuales **NO** cambian.
- Sin embargo, a veces resulta muy útil **modificar**, en una función, variables de los **argumentos actuales**. Esto se consigue usando el Paso de Argumentos POR REFERENCIA**:**
  - Esto es otra forma de conseguir que una función devuelva valores (aparte de usando `return`).
  - Además, de esta forma una función puede devolver tantos valores como se deseen (cada uno en un argumento distinto).
  - **En lenguaje C, TODOS los pasos de argumentos son por VALOR.**
  - Se llama **Paso de Argumentos POR REFERENCIA** a una técnica que permite a una función modificar variables utilizadas como argumentos actuales.

9. Paso de Argumentos por Referencia

- **TÉCNICA** para usar un **paso de argumentos por REFERENCIA en C estándar:**
  - **1. En la Definición de la Función:**
    - Usar el **Operador de Indirección** `*` (asterisco) delante del argumento formal correspondiente.
    - Se usará el **operador** `*` cada vez que la variable del argumento formal sea utilizada, tanto en la declaración de la variable (entre los paréntesis de la función) como en el cuerpo de la función.
  - **2. En la Llamada a la Función:**
    - Usar el **Operador de Dirección** `&` (*ampersand*) delante de la variable para la que se desea el paso de arg. por referencia.
    - Recuerde que el **operador** `&` indica, por tanto, que dicha variable puede verse **modificada** por la función.

9. Paso de Argumentos por Referencia

```
#include<stdio.h>
void f2 (int *x, char *ch){
  *x = *x + 5;
  if (*ch == 'A')
     *ch = 'P';
  else
     *ch = 'K';
}
int main(){
  int x, y=1;
  char letra='A';
  printf("\n- Introduzca un número: ");
  scanf("%i",&x);
  y = y + x;
  f2(&y,&letra);
  printf("\n- Valores: %i, %i y %c.", x, y, letra);
  f2(&x,&letra);
  printf("\n- Valores: %i, %i y %c.", x, y, letra);
  return 0;
}
```

Ejemplo de Ejecución:

```
- Introduzca un número: 3

- Valores: 3, 9 y P.
- Valores: 8, 9 y K.
```


9. Paso de Argumentos por Referencia

- **En C++, la TÉCNICA de paso de argumentos por REFERENCIA se simplifica:**
  - **1. En la Definición de la Función:**
    - Sencillamente hay que marcar qué argumentos tendrán un paso de argumentos por referencia.
    - Esto se hace **añadiendo al tipo del argumento el signo &.**
  - **2. En la Llamada a la Función:**
    - El argumento actual en un paso de arg. por referencia se usa normalmente (como en un paso por valor).
    - En un paso de arg. por referencia el argumento actual debe ser obligatoriamente una variable.

9. Paso de Argumentos por Referencia

```
#include<stdio.h>
void f2 (int& x, char& ch){
  x = x + 5;
  if (ch == 'A')
     ch = 'P';
  else
     ch = 'K';
}
int main(){
  int x, y=1;
  char letra='A';
  printf("\n- Introduzca un número: ");
  scanf("%i",&x);
  y = y + x;
  f2(y,letra);
  printf("\n- Valores: %i, %i y %c.", x, y, letra);
  f2(x,letra);
  printf("\n- Valores: %i, %i y %c.", x, y, letra);
  return 0;
}
```

Dos pasos de arg. por referencia

Ejemplo de Ejecución:

```
- Introduzca un número: 3

- Valores: 3, 9 y P.
- Valores: 8, 9 y K.
```