Programas

Estructura de un programa simple

Los programas más simples escritos en lenguajes imperativos suelen realizar tres tareas de forma secuencial:

- Entrada de datos
- Procesamiento de los datos
- Salida de resultados

La función main

El punto de entrada de un programa en C es la función `main`:

```
int main (int argc, char *argv[])
{
    Declaraciones y sentencias escritas en C

    return 0;
}
```

- Las llaves `{}` delimitan bloques de código en C (conjuntos de declaraciones y sentencias).
- La ejecución de un programa escrito en C comienza invocando a la función `main()`.

El preprocesador de C

#include

Inclusión de ficheros de cabecera

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

#define

Definición de constantes simbólicas y macros

```
#define CONSTANTE expresión
```

```
#define MAX(a,b) ( ((a)>(b))? (a):(b) )
```

#ifndef … #endif

Usado en ficheros de cabecera para no incluir lo mismo dos veces

```
#ifndef __XXX__
#define __XXX__

...

#endif
```

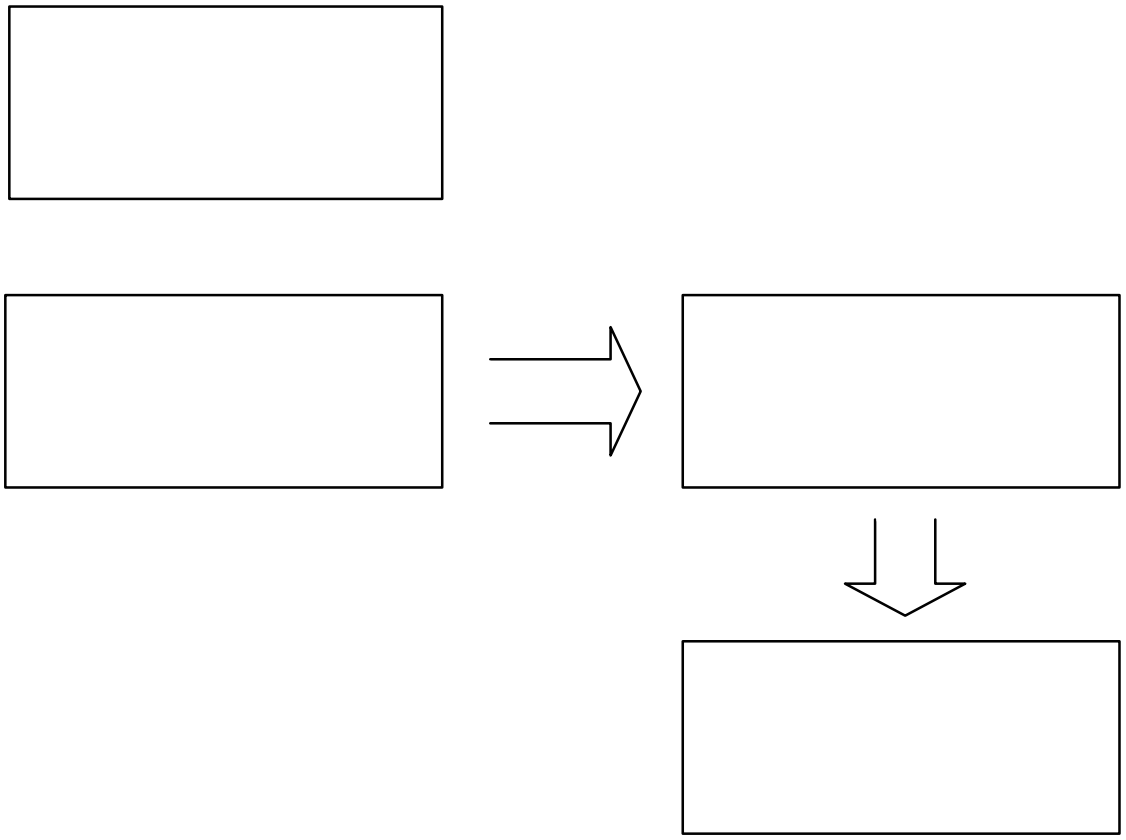
Estructura general de un fichero de código en C

`#includes` de ficheros de cabecera (bibliotecas)

`#defines` (constantes simbólicas y macros)

Declaración e implementación de funciones

Funcionamiento del preprocesador



Fichero de entrada al compilador

Operaciones de entrada/salida

Salida por pantalla con la función `printf`

La función `printf`, que forma parte de la biblioteca estándar de funciones de entrada/salida `<stdio.h>`, nos permite mostrar mensajes de texto en la pantalla cuando ejecutamos un programa:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
  ...
  printf("Mi programa v1.0");
  ...
}
```

Mostrar datos en función de su tipo

```
char c;
...
printf ("%c", c);
```

```
float f;
...
printf ("%f", f);
```

```
int i;
...
printf ("%d", i);
```

```
double d;
...
printf ("%lf", d);
```

```
long x;
...
printf ("%ld", x);
```

```
long double r;
...
printf ("%Lf", r);
```

Mostrar mensajes en distintas líneas

```
printf ("Una línea de texto.\n");
printf ("Otra línea de texto.\n");
```

Entrada desde el teclado con la función `scanf`

La función `scanf`, de la biblioteca estándar de funciones de entrada/salida `<stdio.h>`, nos permite leer datos desde el teclado para suministrarle datos de entrada a un programa:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
  ...
  scanf("%d",&dato);
  ...
}
```

Leer datos en función de su tipo

```
char c;
...
scanf ("%c", &c);
```

```
float f;
...
scanf ("%f", &f);
```

```
int i;
...
scanf ("%d", &i);
```

```
double d;
...
scanf ("%lf", &d);
```

```
long x;
...
scanf ("%ld", &x);
```

```
long double r;
...
scanf ("%Lf", &r);
```

Operaciones de entrada/salida

`printf(`"*plantilla de formato*"`,` *lista de variables*`);`

`scanf(`"*plantilla de formato*"`,` *lista de referencias a variables*`);`

Salida de datos con formato

Al representar un número o una cadena,
podemos especificar cómo deseamos hacerlo:

Sintaxis general	**`%-n.mX`**

- El número `n` indica el número de caracteres que se utilizarán como mínimo para reprentar el dato (número total de dígitos en el caso de los números, tanto enteros como reales).

- El número `.m` indica el número máximo de caracteres que se utilizarán para representar el dato. En el caso de los números reales, indica el número de decimales que se mostrarán.

- El signo menos (`-`) es opcional y lo usaremos para indicar que el texto ha de justificarse a la izquierda.

- `X` indica el tipo del dato:

`%d`	Número entero en decimal
`%x`	Número entero en hexadecimal
`%o`	Número entero en octal
`%c`	Carácter
`%s`	Cadena de caracteres
`%f`	Número real
`%e`	Número real en notación científica

Ejemplos

```
printf( "|%s|", "Hola mundo" )          |Hola mundo|
printf( "|%20s|", "Hola mundo" )        |          Hola mundo|
printf( "|%-20s|", "Hola mundo" )       |Hola mundo          |
printf( "|%20.4s|", "Hola mundo" )      |                Hola|
printf( "|%-20.4s|", "Hola mundo" )     |Hola                |

printf( "|%c|", 'a' )                   |a|
printf( "|%c|", 97  )                   |a|
printf( "|%c|", 'a' + 3 )               |d|


printf( "|%d|", 23423 )                 |23423|
printf( "|%10d|", 23423 )               |     23423|
printf( "|%-10d|", 23423 )              |23423     |
printf( "|%010d|", 23423 )             |0000023423|

printf( "|%10i|", 29387 )              |     29387|
printf( "|%10o|", 29387 )              |     71313|
printf( "|%10x|", 29387 )              |      72cb|

printf( "|%f|", 423.2348 )             |423.234800|
printf( "|%12f|", 423.2348 )           |  423.234800|
printf( "|%12.0f|", 423.2348 )         |         423|
printf( "|%12.2f|", 423.2348 )         |      423.23|
printf( "|%.2f|", 423.2348 )           |423.23|
printf( "|%e|", 423.2348 )             |4.232348e+02|
```

Ejemplo

Longitud de la hipotenusa de un triángulo rectángulo

```
/*
   Hipotenusa de un triángulo rectángulo
   calculada según el teorema de Pitágoras
*/

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

int main ()
{
   // Declaraciones

   float lado1, lado2, hipotenusa;

   // Entrada de datos

   printf(" Cálculo de la hipotenusa \n");
   printf("de un triángulo rectángulo\n");
   printf("--------------------------\n");

   printf("Primer lado: ");
   scanf("%f", &lado1);
   printf("Segundo lado: ");
   scanf("%f", &lado2);

   // Cálculos

   hipotenusa = sqrt(lado1*lado1 + lado2*lado2);

   // Salida de resultados

   printf ("La hipotenusa mide %f.\n", hipotenusa);

   return 0;
}
```

Ejemplo

Programa para comprobar si un año es bisiesto o no

Un año es bisiesto si es divisible por 4 pero no por 100,
o bien es divisible por 400.

```
#include <stdio.h>

int main ()
{
   // Declaración de variables

   int year;
   int bisiesto;

   // Entrada de datos

   printf("Introduzca un año: ");
   scanf("%d", &year);

   // Cálculos

   bisiesto = ((year%4==0) && (year%100!=0))
              || (year%400==0);

   // Salida de resultados

   if (bisiesto) {
      printf ("El año es bisiesto.");
   } else {
      printf ("El año no es bisiesto.");
   }

   return 0;
}
```

NOTA: La sentencia `if` nos permite controlar la ejecución de un conjunto de sentencias en función de que se cumpla una condición.

Ejemplo

Cuota de una hipoteca

$$interésMensual\ (\delta) = interésAnual / 12$$

$$cuotaMensual = \frac{cantidad \times \delta}{1 - \frac{1}{(1+\delta)^{años \times 12}}}$$

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

int main ()
{
    // Declaración de variables

    double cantidad; // en euros
    double interes;  // en porcentaje (anual)
    int    tiempo;   // en años
    double cuota;    // en euros (mensual)

    double interesMensual; // en tanto por uno

    // Entrada de datos

    printf("Importe de la hipoteca (€): ");
    scanf("%lf", &cantidad);

    printf("Tipo de interés (%): ");
    scanf("%lf", &interes);

    printf("Período de amortización (años)");
    scanf("%d", &tiempo);

    // Cálculo de la cuota mensual

    interesMensual = interes/(12*100);

    cuota = (cantidad*interesMensual)
      / (1.0 - 1/pow(1+interesMensual,tiempo*12));

    // Resultado

    printf("Cuota mensual: %.2lf €", cuota);

    return 0;
}
```

Documentación del código

Comentarios

Los comentarios sirven para incluir aclaraciones en el código.

ANSI C permite dos tipos de comentarios:

```
// Comentarios de una línea
/* Comentarios de varias líneas */
```

- Es bueno incluir comentarios que expliquen lo que hace el programa y sus características claves (p.ej. autor, fecha, algoritmos utilizados, estructuras de datos, peculiaridades…).

```
// Cálculo del MCD
// usando el algoritmo de Euclides
// © Fernando Berzal, 2004
```

- Los comentarios nunca han de limitarse a decir en lenguaje natural lo que ya está escrito en el código: Jamás se utilizarán para "parafrasear" el código y repetir lo que es obvio.

```
✗ i++;        // Incrementa el contador
```

- Los comentarios han de aclarar; esto es, ayudar al lector en las partes difíciles (y no confundirle). Si es posible, escriba código fácil de entender por sí mismo: cuanto mejor lo haga, menos comentarios necesitará.

```
✗ int mes;   // Mes
✓ int mes;   // Mes del año (1..12)


✗ ax = 0x723;    /* RIP L.v.B. */
```

NB: Beethoven murió en 1827 (0x723 en hexadecimal).

Sangrías

Conviene utilizar espacios en blanco o separadores para delimitar el ámbito de las estructuras de control de nuestros programas.

Líneas en blanco

Para delimitar claramente los distintos segmentos de código en nuestros programas dejaremos líneas en blanco entre ellos.

Identificadores

Los identificadores deben ser descriptivos (reflejar su significado).

- ✗ `p, i, s...`
- ✓ `precio, izquierda, suma...`

Declaraciones

- Usualmente, declararemos una única variable por línea.
- Nunca mezclaremos en una misma línea la declaración de variables que sean de distintos tipos o que se utilicen en el programa para distintos fines.

Constantes

- Se considera una mala costumbre incluir literales de tipo numérico ("números mágicos") en medio del código.
- Se prefiere la definición de constantes simbólicas (mediante la directiva `#define` del preprocesador de C).

Expresiones

- Uso de paréntesis: Aunque las normas de precedencia de los operadores vienen definidas en el lenguaje, no abusaremos de ellas. Siempre resulta más fácil interpretar una expresión si ésta tiene los paréntesis apropiados. Además, éstos eliminan cualquier tipo de ambigüedad.

- Uso de espacios en blanco: Resulta más fácil leer una expresión con espacios que separen los distintos operadores y operandos involucrados en la expresión.

```
a%x*c/b-1     →   ( (a%x) * c ) / b - 1
```

- Expresiones booleanas: Es aconsejable escribirlas como se dirían en voz alta.

```
!(bloque<actual)     →   ( bloque >= actual )
```

- Expresiones complejas:
  Es aconsejable dividirlas para mejorar su legibilidad

- Claridad:
  Siempre buscaremos la forma más simple de escribir una expresión.

```
✗ key = key >> ( bits - ((bits>>3)<<3));

✓ key >>= bits & 0x7;
```

- Conversiones de tipo (castings):
  Evitaremos las conversiones implícitas de tipo.
  Cuando queramos realizar una conversión de tipo,
  lo indicaremos explícitamente.

```
i = (int) f;
```

- Siempre se han de evitar los efectos colaterales (modificaciones no deseadas pueden afectar a la ejecución del programa sin que nos demos cuenta de ello).

IDEA CLAVE

Escribimos código para que lo puedan leer otras personas,
no sólo para que lo traduzca el compilador (si no fuese así,
podríamos seguir escribiendo nuestros programas en binario).

- No comente el código "malo" (uso de construcciones extrañas, expresiones confusas, sentencias poco legibles...): Reescríbalo.

- No contradiga al código: Los comentarios suelen coincidir con el código cuando se escriben, pero a medida que se corrigen errores y el programa evoluciona, los comentarios suelen dejarse en su forma original y aparecen discrepancias. Si cambia el código, asegúrese de que los comentarios sigan siendo correctos.

El código bien escrito
es más fácil de leer, entender y mantener
(además, seguramente tiene menos errores)

Errores de programación

Errores sintácticos

Errores detectados por el compilador en tiempo de compilación.

Errores semánticos

Sólo se detectan en tiempo de ejecución: Causan que el programa finalice inesperadamente su ejecución (p.ej. división por cero) o que el programa proporcione resultados incorrectos.