Introducción a la programación en C

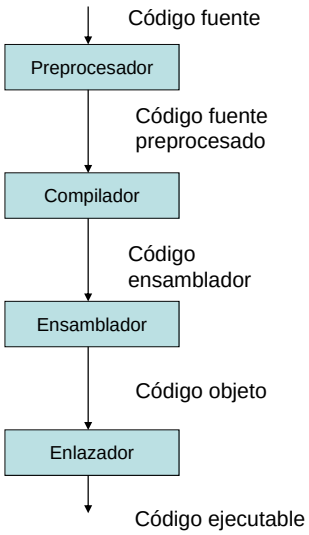
- **Compilador gcc**:
  - gcc (GNU C compiler): Colección de compiladores GNU.
  - gcc hace referencia a una colección de compiladores creados por el proyecto GNU (GNU es un acrónimo recursivo: GNU is Not Unix).
  - El proyecto GNU surgió con el objetivo de crear un sistema operativo libre compatible con Unix. (Linux, el sistema operativo libre de tipo Unix, se conoce como también como GNU/Linux).
  - gcc es el compilador más usado en ordenadores con sistema operativo de tipo Unix.
  - gcc inicialmente solo compilaba C, aunque posteriormente se extendió para compilar C++, Fortran, Ada,…

1

Etapas de la compilación:

La traducción de un programa escrito en un lenguaje C a código máquina comprende 4 etapas:

- Preprocesado: se eliminan los comentarios y se interpretan las directivas de preprocesador.
- Compilación: se genera código ensamblador.
- Ensamblado: se genera código máquina no ejecutable denominado código objeto que contiene referencias a funciones de otras bibliotecas.
- Enlazado: se combinan los códigos objeto para crear el código máquina (ya ejecutable).



2

– 4 etapas en la compilación:
  - Preprocesado.
  - Compilación.
  - Ensamblado.
  - Enlazado.

– Para cada fichero de entrada, su extensión determina el tipo de compilación a hacer:
  .c: fichero fuente en lenguaje C.
  .i: fichero preprocesado.
  .s: fichero en lenguaje ensamblador.
  .o: fichero objeto.

3

- Sintaxis del comando:
  gcc [-opciones] lista_de_ficheros

  Opciones:

  -Wall: activa los avisos más comunes (warnings)

  -o nombre_fichero: crea el ejecutable en nombre_fichero en vez de en “a.exe”.

  -E: detiene la compilación tras la etapa de preprocesado.

  -S: detiene la compilación tras la etapa de compilación. Genera fichero en lenguaje ensamblador.

  -c: detiene la compilación tras la etapa de ensamblado. Genera fichero objeto.

4

– Primer programa en C:
  - Programa cuya ejecución hace que se visualice por pantalla el mensaje "Hola mundo".

```
#include <stdio.h>
/* El primer programa en C: visualiza por pantalla un mensaje al ejecutarse */

/* La función main inicia la ejecución */

int main(void)
{
   printf("Hola mundo\n");
   return 0;
}
/* fin de la función main */
```

5

gcc –Wall holamundo.c → compila el fichero fuente *holamundo.c* y llama al fichero ejecutable con el nombre *a.exe* (nombre por defecto si no se indica uno).

a → ejecuta el fichero ejecutable a.exe

Para dar al fichero ejecutable un nombre concreto, por ejemplo, *holamundo*:

gcc holamundo.c –o holamundo
gcc –o holamundo holamundo.c
} ambas opciones válidas

holamundo → ejecuta el fichero ejecutable holamundo.exe

gcc –E holamundo.c → se realiza la etapa de preprocesado y se visualiza por pantalla el fichero preprocesado

gcc –E holamundo.c >holamundo.i → se redirecciona la salida de la etapa de preprocesado al fichero holamundo.i

6

gcc –S holamundo.c → se detiene la compilación después la etapa de compilación (tras realizarse las etapas de preprocesado y de compilación) y se crea el fichero *holamundo.s* con código en lenguaje ensamblador resultado de esas 2 etapas.

gcc –c holamundo.c → detiene la compilación después la etapa de ensamblado. Genera fichero con código objeto *holamundo.o*

7

Lenguaje de programación C

- El lenguaje de programación C fue creado por Dennis Ritchie en 1972 en los laboratorios Bell para desarrollar el sistema operativo Unix.
- La idea básica de Ritchie era crear un lenguaje de propósito general que realizara muchas de las tareas reservadas anteriormente a los programas escritos en lenguajes ensambladores y con el que resultara fácil programar.
- El lenguaje C obtuvo un gran éxito gracias a la combinación de las ventajas de los lenguajes compilados y los ensambladores.
- Características de C:
  - Es un lenguaje de propósito general que puede ser utilizado para la programación de una gran variedad de aplicaciones. Se puede utilizar para desarrollar sistemas operativos, compiladores, sistemas en tiempo real y aplicaciones de comunicaciones.

8

- Es un lenguaje que comparte las ventajas de los lenguajes ensambladores (lenguajes de bajo nivel: permiten un dominio completo de la máquina) y las ventajas de los lenguajes de alto nivel.

- Es un lenguaje que hace posible una programación modular. Los programas pueden escribirse en módulos independientes, almacenando cada módulo en un fichero aparte y compilándolos separadamente. Luego, estos módulos compilados (códigos objeto) pueden enlazarse para crear el código máquina ejecutable.

- Es un lenguaje conciso que da lugar a un código muy compacto.

- Aparte de la biblioteca estándar, que es la que vamos a utilizar, se han creado muchas bibliotecas del lenguaje C que soportan aplicaciones de bases de datos, gráficos, comunicaciones,…

9

– Primer programa en C:
  - Programa cuya ejecución hace que se visualice por pantalla el mensaje “Hola mundo”.

```
#include <stdio.h>
/* El primer programa en C: visualiza por pantalla un mensaje al ejecutarse */

/* La función main inicia la ejecución */

int main(void)
{
   printf(“Hola mundo\n”);
   return 0;
}
/* fin de la función main */
```

10

– Vamos a ver las distintas partes del programa:

#include <stdio.h>

- Es una directiva de preprocesador: modifican el texto del programa en la fase de preprocesado de la compilación.
- Esta directiva (*#include <stdio.h>)* está diciendo que se incluya en el programa el fichero de cabecera *stdio.h,* que contiene la declaración de las funciones de entrada y salida de la biblioteca estándar de C (en la que está la función printf).
- Por ejemplo, en el fichero *math.h*, está la declaración de las funciones de biblioteca matemática
- También sirven las directivas de preprocesador para definir constantes simbólicas:
  - Por ejemplo, *#define PI 3.1416*, hace que en la fase de preprocesado, cada vez que se encuentre *PI* en el código (en C se suelen escribir las constantes en mayúsculas), se sustituya por *3.1416*

    *a=2*PI*r* (tras preprocesado) → *a=2*3.1416*r*

11

Comentarios:

- Son anotaciones usadas para clarificar el programa (documentación interna).
- Para abrir un comentario se escribe /* y para cerrarlo */
- En la fase de preprocesado de la compilación se eliminan los comentarios.

int main(void)

- La función *main* (su definición) marca el punto donde todo programa en C comienza su ejecución. Es obligatorio que en todos los programas haya una función *main*.
- Los programas en C tienen una o más funciones, una de las cuales debe ser la función *main*.

  int main(void) equivale a poner *main()*

  int -> la función *main* devuelve un entero
  void -> la función *main* no tiene argumentos de entrada.

- Cada función tiene entre llaves ({…}) su contenido o cuerpo.

12

printf("Hola mundo\n");

- *Es una función de salida de la biblioteca estándar.*
- *Se usa para escribir información con formato por la salida estándar, que es normalmente la pantalla del ordenador.*
- *Toda instrucción debe acabar con un punto y coma (;)*
- *Es este caso, imprime por pantalla la cadena de caracteres incluidos entre comillas (" ")*
- *La barra inclinada a la izquierda (contrabarra) es un carácter de escape, que sirve para introducir secuencias de escape.*
- *Ejemplos de secuencias de escape:*
  - \n → nueva línea
  - \t → tabulación horizontal
  - \\ → inserta una contrabarra y así no es interpretada como carácter de escape.
  - \" → inserta unas comillas dentro del texto entre comillas y así no termina el texto a imprimir

13

return 0;

- Es la manera usual de finalizar la ejecución de una función. La ejecución de la instrucción *return 0;* hace que el programa salga de la función *main*, devolviendo un valor de 0. Al finalizar la ejecución de la función *main*, se devuelve el control al sistema operativo.
- Muy recomendable utilizar sangrías (código sangrado), que ayudan a leer y entender los programas.

14

– Programa que suma dos números enteros:

```
#include <stdio.h>/* la función main inicia la ejecución del programa */

int main(void)
{
   int entero1; /* primer número introducido por el usuario  */
   int entero2; /* segundo número introducido por el usuario */
   int suma;    /* variable donde se almacenara la suma */

   printf( "Introduzca el primer entero: " );
   scanf( "%d", &entero1 );         /* lee un entero */

   printf( "Introduzca el segundo entero: " );
   scanf( "%d", &entero2 );         /* lee un entero */

   suma = entero1 + entero2; /* asigna el resultado a suma */

   printf( "La suma es %d\n", suma ); /* imprime la suma */

   return 0;
}
```

15

Variables:

int entero1;
int entero2;
int suma;

- Tenemos 3 declaraciones de variables, que se van a utilizar en el programa *suma.c* para almacenar valores de entrada y resultados de las operaciones.
- En la declaración de una variable primero aparece el tipo de dato que se almacena en la variable y después el nombre de la variable (finalizando con ;).
- Cada vez que se declara una variable se la asocian 3 atributos:
  - Un tipo (Ej.: *int* → tipo entero).
  - Un nombre (Ej.: *entero1*, *entero2*, *suma*).
  - Un espacio en memoria que almacena su valor.
- Una variable puede considerarse un espacio en memoria de tamaño adecuado para que pueda almacenarse su valor.
  Ej.: *int num=12;*
  - » Instrucción que indica “Reservar un espacio en memoria en el que quepa un entero, llamar a ese espacio num y darle el valor 12”
  - » En este caso la variable num está declarada e inicializada (en la declaración de la variable, la inicialización es opcional).

16

– En C las variables se tienen que declarar mediante un tipo, tienen que ser de un tipo.

– Tipos de datos escalares (clasificación de las variables por el tipo de dato que guardan):

  • *int* → número entero:
    • Si ocupa 2 bytes (16 bits) tendrá el rango: -32768 a 32767 (-2^{15} a $2^{15}-1$)
    • Si ocupa 4 bytes (32 bits) tendrá el rango: -2147483648 a 2147483647 (-2^{31} a $2^{31}-1$)

  • *flo*at → número real (coma flotante de simple precisión). Ocupa 4 bytes.
    • rango para valores positivos: $\sim 10^{\pm 38}$

  • *double* → número real (coma flotante de doble precisión). Ocupa 8 bytes.
    • rango para valores positivos: $\sim 10^{\pm 308}$

  • *char* → carácter. Ocupa 1 byte: para almacenar un carácter del juego de caracteres del ordenador.
    • rango: 0 a 255

17

– Los tipos de datos básicos vistos pueden tener modificadores que les preceden, como *short, long, unsigned, signed*.

– Se usa un modificador para alterar el significado de un tipo base para que encaje con las diversas necesidades.

  • Ej.: unsigned int x;

– Si no se especifica el tipo en una declaración de una variable sino solo el modificador, el tipo que se especifica por defecto es int.

  • Son equivalentes:
    unsigned int x;
    unsigned x;

    • El rango de x, si el entero ocupa 4 bytes, al ser solo para números sin signo, positivos, pasa de:
      -2^{31} a $2^{31}-1$
      a
      0 a $2^{32}-1$ (0 a 4294967295)

18

– Como nombre de variable vale:

- Cualquier identificador válido (combinación de letras, dígitos y guiones bajos) tal que:
  - No comience con un dígito. Ej.: 1variable no es un nombre válido
  - No sea una palabra reservada del lenguaje. Ej.: *printf*, *int*
  - Su longitud sea menor o igual a 31 caracteres.
- Se distingue entre mayúsculas y minúsculas.
  Ej.: Suma, SUMA y suMa son 3 nombres de variables distintos.
- Es recomendable elegir los nombres de las variables tal que tengan relación con el valor que guardan.
  Ej.: La variable con nombre *suma* para recoger el valor de una suma de otras variables.

19

– Función *scanf*:

- Función que permite leer información con formato de la entrada estándar (normalmente el teclado).
- Es una función de entrada de la biblioteca estándar ANSI C (requiere incluir el fichero de cabecera *stdio.h*).
- La biblioteca estándar ANSI C está formada por funciones agrupadas en bibliotecas:
  - Funciones de entrada y salida (*scanf*, *printf*) están en la biblioteca estándar. Si usamos alguna de ellas, hay que incluir el fichero de cabecera *stdio.h*
  - Funciones matemáticas están en la biblioteca matemática. Si usamos alguna de ellas, tendremos que incluir el fichero de cabecera *math.h*

20

- Los argumentos en general de la función *scanf* al llamarla son:

scanf(cadena de control del formato, &variable1, &variable2,…);

- La cadena de control del formato contiene entre comillas dobles los tipos de datos que debe introducir el usuario por teclado, precedidos cada uno de los tipos por %.

- Cada conjunto de % y tipo de dato se denomina código de formato.

  Códigos de formato:
  %d → int (número entero)
  %f → float (número real de simple precisión)
  %lf → double (número real de doble precisión)
  %c → carácter
  %s → cadena de caracteres

- *&variable1, &variable2* hacen referencia a las direcciones de memoria de las variables, donde se van a almacenar los valores de esas variables que introducimos por teclado.

21

- En el programa:

scanf("%d",&entero1);

- scanf con 2 argumentos:
  - El primer argumento: cadena de control del formato, indica que el usuario debe introducir por teclado un dato de tipo entero.
  - El segundo argumento indica que el valor introducido se va a almacenar en la dirección de memoria de la variable *entero1*.

- *scanf* tiene al menos 2 argumentos.

22

- printf tiene al menos 1 argumento, que es el caso que hemos visto:
  printf("Hola mundo");

  pero puede tener más, como en el programa *suma.c*:
  printf("La suma es %d\n",suma);

- Los argumentos en general de la función *printf* al llamarla son:
  printf(cadena de control del formato, variable1, variable2,…):

- La cadena de control del formato tiene la cadena de caracteres a visualizar en pantalla y los códigos de formato de cada una de las variables a visualizar.

- Podríamos tener en este programa:
  printf("La suma de %d y de %d es %d\n",entero1,entero2,suma);

23

– Operadores aritméticos:
  - Permiten hacer operaciones aritméticas sobre las variables.
    + → suma
    - → resta
    * → multiplicación
    / → división
    % → módulo

  - Precedencia o prioridad de las operaciones (orden en el que se ejecutan las operaciones si no hay paréntesis, que tiene mayor precedencia):

    1) *, / , %
    2) +, -

    Entre los operadores aritméticos de igual precedencia, se ejecutan las operaciones de izquierda a derecha.

    Ej.: a*b%c es distinto de a*(b%c)

24

– Operador de asignación (=):
  - Atribuyen a una variable el resultado de una expresión o el valor de otra variable

    Ej.: a=b+c
    entero=entero+5 → entero vale su valor anterior más 5

25

Lectura y escritura de un fichero de texto

int entero;
scanf("%d",&entero); → lectura de entero por teclado

fscanf(fichero,"%d",&entero); → lectura de entero de un fichero de texto

printf("Entero vale %d\n",entero); → escritura de entero por teclado

fprintf(fichero, "Entero vale %d\n",entero); → escritura de entero en fichero de texto

26

– Programa que suma dos números enteros trabajando con ficheros:

```
#include <stdio.h>/* la función main inicia la ejecución del programa */

int main(void)
{
        int entero1; /* primer número leido del fichero  */
        int entero2; /* segundo número leido del fichero*/
        int suma;    /* variable donde se almacenara la suma */
        FILE *punterofichero1;
        FILE *punterofichero2;
        punterofichero1=fopen("fichero.txt","rt");
        punterofichero2=fopen("fichero2.txt","wt");

        printf("Lee el primer entero del fichero: " );
        fscanf( punterofichero1,"%d", &entero1 );         /* lee un entero */
        printf("%d\n",entero1);

        printf("Lee el segundo entero del fichero: " );
        fscanf( punterofichero1,"%d", &entero2 );         /* lee un entero */
        printf("%d\n",entero2);

        suma = entero1 + entero2; /* asigna el resultado a suma */

        printf("La suma es %d\n", suma ); /* imprime la suma */
        fprintf(punterofichero2,"La suma es %d\n", suma );
        fclose(punterofichero1);
        fclose(punterofichero2);

        return 0;
}
```

27