C99 y `gcc`

Por defecto, `gcc` acepta programas escritos en C89 con extensiones introducidas por GNU (el grupo de desarrolladores de muchas herramientas de Linux). Muchas de esas extensiones de GNU forman ya parte de C99, así que `gcc` es, por defecto, el compilador de un lenguaje intermedio entre C89 y C99. Si en algún momento da un aviso indicando que no puede compilar algún programa porque usa características propias del C99 no disponibles por defecto, puedes forzarle a compilar en «modo C99» así:

```
gcc programa.c -std=c99 -o programa
```

Has de saber, no obstante, que `gcc` aún no soporta el 100% de C99 (aunque sí todo lo que te explicamos en este texto).

El compilador `gcc` acepta muchas otras variantes de C. Puedes forzarle a aceptar una en particular «asignando» a la opción `-std` el valor `c89`, `c99`, `gnu89` o `gnu99`.

1.2. Traduciendo de Python a C: una guía rápida

Empezaremos por presentar de forma concisa cómo traducir la mayor parte de los programas Python que aprendimos a escribir en los capítulos 3 y 4 del primer volumen a programas equivalentes en C. En secciones posteriores entraremos en detalle y nos dedicaremos a estudiar las muchas posibilidades que ofrece C a la hora de seleccionar tipos de datos, presentar información con sentencias de impresión en pantalla, etc.

1. Los programas (sencillos) presentan, generalmente, este aspecto:

```
#include <stdio.h>

Posiblemente otros «#include»

int main(void)
{
  Programa principal.

  return 0;
}
```

Hay, pues, dos zonas: una inicial cuyas líneas empiezan por **#include** (equivalentes a las sentencias **import** de Python) y una segunda que empieza con una línea «**int** *main*`(`**void**`)`» y comprende las sentencias del programa principal mas una línea «**return** `0;`», encerradas todas ellas entre llaves (`{` y `}`).

De ahora en adelante, todo texto comprendido entre llaves recibirá el nombre de *bloque*.

2. Toda variable debe declararse antes de ser usada. La declaración de la variable consiste en escribir el nombre de su tipo (**int** para enteros y **float** para flotantes)[3] seguida del identificador de la variable y un punto y coma. Por ejemplo, si vamos a usar una variable entera con identificador a y una variable flotante con identificador b, nuestro programa las declarará así:

```
#include <stdio.h>

int main(void)
{
  int a;
  float b;

  Sentencias donde se usan las variables.

  return 0;
}
```

[3] Recuerda que no estudiaremos las variables de tipo cadena hasta el próximo capítulo.

No es obligatorio que la declaración de las variables tenga lugar justo al principio del bloque que hay debajo de la línea «**int** *main*(**void**)», pero sí conveniente.[4]

Si tenemos que declarar dos o más variables del mismo tipo, podemos hacerlo en una misma línea separando los identificadores con comas. Por ejemplo, si las variables x, y y z son todas de tipo **float**, podemos recurrir a esta forma compacta de declaración:

```
#include <stdio.h>

int main(void)
{
  float x, y, z;

  ...

  return 0;
}
```

3. Las sentencias de asignación C son similares a las sentencias de asignación Python: a mano izquierda del símbolo igual (=) se indica la variable a la que se va a asignar el valor que resulta de evaluar la expresión que hay a mano derecha. Cada sentencia de asignación debe finalizar con punto y coma.

```
#include <stdio.h>

int main(void)
{
  int a;
  float b;

  a = 2;
  b = 0.2;

  return 0;
}
```

Como puedes ver, los números enteros y flotantes se representan igual que en Python.

4. Las expresiones se forman con los mismos operadores que aprendimos en Python. Bueno, hay un par de diferencias:

   - Los operadores Python **and**, **or** y **not** se escriben en C, respectivamente, con `&&`, `||` y `!`;
   - No hay operador de exponenciación (que en Python era `**`).
   - Hay operadores para la conversión de tipos. Si en Python escribíamos *float*(*x*) para convertir el valor de x a flotante, en C escribiremos **(float)** x para expresar lo mismo. Fíjate en cómo se disponen los paréntesis: los operadores de conversión de tipos son de la forma **(tipo)**.

```
#include <stdio.h>

int main(void)
{
  int a;
  float b;

  a = 13 % 2;
  b = 2.0 / (1.0 + 2 - (a + 1));

  return 0;
}
```

[4] En versiones de C anteriores a C99 sí era obligatorio que las declaraciones se hicieran al principio de un bloque. C99 permite declarar una variable en cualquier punto del programa, siempre que éste sea anterior al primer uso de la misma.

Las reglas de asociatividad y precedencia de los operadores son casi las mismas que aprendimos en Python. Hay más operadores en C y los estudiaremos más adelante.

5. Para mostrar resultados por pantalla se usa la función *printf*. La función recibe uno o más argumentos separados por comas:

   - primero, una cadena con formato, es decir, con marcas de la forma `%d` para representar enteros y marcas `%f` para representar flotantes (en los que podemos usar modificadores para, por ejemplo, controlar la cantidad de espacios que ocupará el valor o la cantidad de cifras decimales de un número flotante);
   - y, a continuación, las expresiones cuyos valores se desea mostrar (debe haber una expresión por cada marca de formato).

escribe.c — escribe.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    float b;
7
8    a = 13 % 2;
9    b = 2.0 / (1.0 + 2 - (a + 1));
10
11   printf("El valor de a es %d y el de b es %f\n", a, b);
12
13   return 0;
14 }
```

La cadena con formato debe ir encerrada entre comillas dobles, no simples. El carácter de retorno de carro (`\n`) es obligatorio si se desea finalizar la impresión con un salto de línea. (Observa que, a diferencia de Python, no hay operador de formato entre la cadena de formato y las expresiones: la cadena de formato se separa de la primera expresión con una simple coma).

Como puedes ver, todas las sentencias de los programas C que estamos presentando finalizan con punto y coma.

6. Para leer datos de teclado has de usar la función *scanf*. Fíjate en este ejemplo:

lee_y_escribe.c — lee_y_escribe.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    float b;
7
8    scanf("%d", &a);
9    scanf("%f", &b);
10
11   printf("El valor de a es %d y el de b es %f\n", a, b);
12
13   return 0;
14 }
```

La línea 8 lee de teclado el valor de un entero y lo almacena en a. La línea 9 lee de teclado el valor de un flotante y lo almacena en b. Observa el uso de marcas de formato en el primer argumento de *scanf*: `%d` señala la lectura de un **int** y `%f` la de un **float**. El símbolo `&` que precede al identificador de la variable en la que se almacena el valor leído es obligatorio para variables de tipo escalar.

Si deseas mostrar por pantalla un texto que proporcione información acerca de lo que el usuario debe introducir, hemos de usar nuevas sentencias *printf*:

lee_mejor_y_escribe.c — lee_mejor_y_escribe.c

```c
#include <stdio.h>

int main(void)
{
  int a;
  float b;

  printf("Introduce un entero a: ");
  scanf("%d", &a);
  printf("Y ahora un flotante b: ");
  scanf("%f", &b);

  printf("El valor de a es %d y el de b es %f\n", a, b);

  return 0;
}
```

7. La sentencia **if** de Python presenta un aspecto similar en C:

si_es_par.c — si_es_par.c

```c
#include <stdio.h>

int main(void)
{
  int a;

  printf("Introduce un entero a: ");
  scanf("%d", &a);

  if (a % 2 == 0) {
    printf("El valor de a es par.\n");
    printf("Es curioso.\n");
  }

  return 0;
}
```

Ten en cuenta que:

- la condición va encerrada obligatoriamente entre paréntesis;
- y el bloque de sentencias cuya ejecución está supeditada a la satisfacción de la condición va encerrado entre llaves (aunque matizaremos esta afirmación más adelante).

Naturalmente, puedes anidar sentencias **if**.

si_es_par_y_positivo.c — si_es_par_y_positivo.c

```c
#include <stdio.h>

int main(void)
{
  int a;

  printf("Introduce un entero a: ");
  scanf("%d", &a);

  if (a % 2 == 0) {
    printf("El valor de a es par.\n");
    if (a > 0) {
      printf("Y, además, es positivo.\n");
    }
  }

```

```
17   return 0;
18 }
```

También hay sentencia **if-else** en C:

par_o_impar.c par_o_impar.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6
7    printf("Introduce un entero a: ");
8    scanf("%d", &a);
9
10   if (a % 2 == 0) {
11     printf("El valor de a es par.\n");
12   }
13   else {
14     printf("El valor de a es impar.\n");
15   }
16
17   return 0;
18 }
```

No hay, sin embargo, sentencia **if-elif**, aunque es fácil obtener el mismo efecto con una sucesión de **if-else if**:

tres_casos.c tres_casos.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6
7    printf("Introduce un entero a: ");
8    scanf("%d", &a);
9
10   if (a > 0) {
11     printf("El valor de a es positivo.\n");
12   }
13   else if (a == 0) {
14     printf("El valor de a es nulo.\n");
15   }
16   else if (a < 0) {
17     printf("El valor de a es negativo.\n");
18   }
19   else {
20     printf("Es imposible mostrar este mensaje.\n");
21   }
22
23   return 0;
24 }
```

8. La sentencia **while** de C es similar a la de Python, pero has de tener en cuenta la obligatoriedad de los paréntesis alrededor de la condición y que las sentencias que se pueden repetir van encerradas entre un par de llaves:

cuenta_atras.c cuenta_atras.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
```

```
5   int a;
6
7   printf("Introduce un entero a: ");
8   scanf("%d", &a);
9
10  while (a > 0) {
11    printf("%d", a);
12    a -= 1;
13  }
14  printf("¡Boom!\n");
15
16  return 0;
17 }
```

9. También puedes usar la sentencia **break** en C:

primo.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a, b;
6
7    printf("Introduce un entero a: ");
8    scanf("%d", &a);
9
10   b = 2;
11   while (b < a) {
12     if (a % b == 0) {
13       break;
14     }
15     b += 1;
16   }
17   if (b == a) {
18     printf("%d es primo.\n", a);
19   }
20   else {
21     printf("%d no es primo.\n", a);
22   }
23
24   return 0;
25 }
```

10. Los módulos C reciben el nombre de *bibliotecas* y se importan con la sentencia **#include**. Ya hemos usado **#include** en la primera línea de todos nuestros programas: **#include** `<stdio.h>`. Gracias a ella hemos importado las funciones de entrada/salida *scanf* y *printf*. No se puede importar una sola función de una biblioteca: debes importar el contenido completo de la biblioteca.

    Las funciones matemáticas pueden importarse del módulo matemático con **#include** `<math.h>` y sus nombres son los mismos que vimos en Python (*sin* para el seno, *cos* para el coseno, etc.).

raiz_cuadrada.c

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(void)
5  {
6    float b;
7
8    printf("Escribe un flotante: ");
9    scanf("%f", &b);
```

```
10
11    if (b >= 0.0) {
12      printf("Su raíz cuadrada es %f.\n",  sqrt(b));
13    }
14    else {
15      printf("No puedo calcular su raíz cuadrada.\n");
16    }
17
18    return 0;
19  }
```

No hay funciones predefinidas en C. Muchas de las que estaban predefinidas en Python pueden usarse en C, pero importándolas de bibliotecas. Por ejemplo, *abs* (valor absoluto) puede importarse del módulo `stdlib.h` (por «standard library», es decir, «biblioteca estándar»).

Las (aproximaciones a las) constantes π y e se pueden importar de la biblioteca matemática, pero sus identificadores son ahora `M_PI` y `M_E`, respectivamente.

No está mal: ya sabes traducir programas Python sencillos a C (aunque no sabemos traducir programas con definiciones de función, ni con variables de tipo cadena, ni con listas, ni con registros, ni con acceso a ficheros...). ¿Qué tal practicar con unos pocos ejercicios?

EJERCICIOS

▶ **2** Traduce a C este programa Python.

```
a = int(raw_input('Dame el primer número: '))
b = int(raw_input('Dame el segundo número: '))

if a >= b:
  maximo = a
else:
  maximo = b

print 'El máximo es', maximo
```

▶ **3** Traduce a C este programa Python.

```
n = int(raw_input('Dame un número: '))
m = int(raw_input('Dame otro número: '))

if n * m == 100:
  print 'El producto %d * %d es igual a 100' % (n, m)
else:
  print 'El producto %d * %d es distinto de 100' % (n, m)
```

▶ **4** Traduce a C este programa Python.

```
from math import sqrt

x1 = float(raw_input("Punto 1, coordenada x: "))
y1 = float(raw_input("Punto 1, coordenada y: "))
x2 = float(raw_input("Punto 2, coordenada x: "))
y2 = float(raw_input("Punto 2, coordenada y: "))
dx = x2 - x1
dy = y2 - y1
distancia = sqrt(dx**2 + dy**2)
print 'La distancia entre los puntos es: ', distancia
```

▶ **5** Traduce a C este programa Python.

```
a = float(raw_input('Valor de a: '))
b = float(raw_input('Valor de b: '))

if a != 0:
  x = -b/a
```