

# RECETA Y TESTING EN C

Cátedra Programación II

Octubre 2016

## 1. Construcción de Programas

Como vimos anteriormente en **Python**, la receta que aplicamos consta de los siguientes pasos:

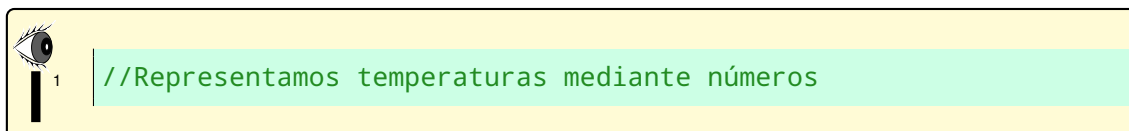
1. diseño de datos;
2. signatura y declaración de proposito.
3. ejemplos;
4. definición de la función;
5. evaluar el código de los ejemplos;
6. realizar modificaciones en caso que el paso anterior genere errores.

Vamos a ver esto, en **C**, aplicado al mismo ejemplo que vimos en *Racket* y *Python* :

Escribir un programa que convierta una temperatura medida en un termómetro Fahrenheit a una temperatura en Celsius.

### 1.1. Item 1: Diseño de datos

*¿ Cómo representamos la información ?*




### 1.2. Item 2: Signatura y declaración de propósito

*La signatura de una función indica qué parámetros recibe (cuántos y de qué tipo), y qué datos retorna. La descripción de propósito indica una breve descripción del ¿Qué hace? la función*

En el caso de los problemas sencillos que abordaremos, deberemos decidir:

- a) cuáles son los datos de entrada que se nos proveen,
- b) cuáles son las salidas que debemos producir, y
- c) cuál es la relación entre todos ellos.


Como vimos, puede pasar que una función en C no tome argumentos o no retorne valores. En el caso de que no tome argumentos



```
1 //Representamos temperaturas mediante números enteros
2 //farCel: Int -> Int
3 //El parámetro representa una temperatura en Fahrenheit y,
4 // se retorna su equivalente en Celsius.
```

### 1.3. Item 3: Ejemplos

*Luego de los pasos anteriores, podemos saber el resultado de la función para algunos valores de entrada*




```
1 //Representamos temperaturas mediante números enteros
2 //farCel: Int -> Int
3 //El parámetro representa una temperatura en Fahrenheit y,
4 // se retorna su equivalente en Celsius.
5 // entrada: 32, salida: 0
6 // entrada: 212, salida: 100
7 // entrada: -40, salida: -40
```

### 1.4. Item 4: Definición del programa

*¡Escribimos código!*

Traducir a un lenguaje de programación el diseño que elegimos en el punto anterior.




```
1 //Representamos temperaturas mediante números enteros
2 //farCel: int -> int
3 //El parámetro representa una temperatura en Fahrenheit y,
4 // se retorna su equivalente en Celsius.
5 // entrada: 32, salida: 0
6 // entrada: 212, salida: 100
7 // entrada: -40, salida: -40
8 int farCel(int f) {
9     return (f-32)*5/9;
10 }
```

## 1.5. Item 5: Evaluar el código en los ejemplos


¿ Qué significa ?

Aplicar el código generado en los ejemplos que se diseñaron y, verificar que los resultados obtenidos coincidan con lo esperado. Para esto, armamos un mail que invoque los casos de ejemplos y mostramos la salida obtenida:



```
1 int main() {  
2     printf("farCel(32) = %d\n", farCel(32));  
3     printf("farCel(212) = %d\n", farCel(212));  
4     printf("farCel(-40) = %d\n", farCel(-40));  
5     return 0;  
6 }
```

Cuya salida, al ejecutarlo va a ser:



```
farCel(32) = 0  
farCel(212) = 100  
farCel(-40) = -40
```

## 1.6. Item 6: Realizar modificaciones en caso de error

¿ Qué significa ?

Encontrar los problemas que se hubieran detectado y solucionarlos.

## 2. Un Ejemplo: Conversión de Unidades Métricas

PROBLEMA 1. Al leer un artículo en una revista que contiene información de longitudes expresadas en millas, pies y pulgadas, correspondientes al sistema anglosajón de unidades no métricas, necesitamos convertir esas distancias a metros. Para ello decidimos escribir un programa que convierta las longitudes del **sistema anglosajón** al **sistema métrico decimal**.

### ***Diseño de datos***

En este caso el problema es sencillo: representamos millas, pies, pulgadas y metros como números flotantes (con coma).

### ***Signatura y declaración de propósito***



```
1 //Representamos millas, pies, pulgadas y metros como números
2 //convierteAMetros: double -> double -> double -> double
3 //El primer parámetro representa una longitud en millas,
4 //el segundo en pies, el tercero en pulgadas y, el valor de
5 //retorno representa su equivalente en metros.
```

## Ejemplos

Para construir los ejemplos, tenemos que encontrar la relación existente entre las unidades que tomamos de entrada y, la salida que vamos a generar. Para esto, buscando en Internet, encontramos la siguiente tabla:

- 1 milla = 1609.344 m
- 1 pie = 0.3048 m
- 1 pulgada=0.0254 m

Por lo que podemos decir que



```
1 //Representamos millas, pies, pulgadas y metros como números
2 //convierteAMetros: double -> double -> double -> double
3 //El primer parámetro representa una longitud en millas,
4 //el segundo en pies, el tercero en pulgadas y, el valor de
5 //retorno representa su equivalente en metros.
6 //entrada: 1, 0, 0; salida: 1609.344
7 //entrada: 0, 1, 0; salida: 0.3048
8 //entrada: 0, 0, 1; salida: 0.0254
9 //entrada: 1, 1, 1; salida: 1609.6742
```

## Definición del programa

Ahora, vamos a escribir el código de la función en C:



```
1 //Representamos millas, pies, pulgadas y metros como números
2 //convierteAMetros: double -> double -> double -> double
3 //El primer parámetro representa una longitud en millas,
4 //el segundo en pies, el tercero en pulgadas y, el valor de
5 //retorno representa su equivalente en metros.
6 //entrada: 1, 0, 0; salida: 1609.344
7 //entrada: 0, 1, 0; salida: 0.3048
8 //entrada: 0, 0, 1; salida: 0.0254
9 //entrada: 1, 1, 1; salida: 1609.6742
```

```
10 double convierteAMetros(double numeroMillas, double numeroPie,↵  
    double numeroPulgadas) {  
11     double metros;  
12     metros = 1609.344 * numeroMillas + 0.3048 * numeroPie + ↵  
        0.0254 * numeroPulgadas;  
13     return metros;  
14 }
```

entendiendo que si una longitud se expresa como  $L$  millas,  $F$  pies y  $P$  pulgadas, su conversión a metros se calculará como:

$$M = 1609.344 \times L + 0.3048 \times F + 0.0254 \times P \quad (1)$$

### Evaluar el código en los ejemplos

Aquí tendríamos que verificar los ejemplos mencionados por lo que el main quedaría así:

```
1 int main() {  
2     printf("convierteAMetros(1,0,0) = %lf\n", convierteAMetros↵  
        (1,0,0));  
3     printf("convierteAMetros(0,1,0) = %lf\n", convierteAMetros↵  
        (0,1,0));  
4     printf("convierteAMetros(0,0,1) = %lf\n", convierteAMetros↵  
        (0,0,1));  
5     printf("convierteAMetros(1,1,1) = %lf\n", convierteAMetros↵  
        (1,1,1));  
6     return 0;  
7 }
```

Cuya salida, al ejecutarlo va a ser:

```
convierteAMetros(0,1,0) = 1609.344000  
convierteAMetros(0,1,0) = 0.304800  
convierteAMetros(0,0,1) = 0.025400  
convierteAMetros(1,1,1) = 1609.674200
```

Con estos ejemplos podemos ver cómo aplicaríamos la receta a cada función que forme parte del programa.

### 3. ¿Cómo testear los ejemplos?

Para automatizar los casos de prueba en **C** vamos a utilizar la librería **assert.h** la cual vamos a tener que incluir como cabecera en el programa:



```
#include <assert.h>
```

la librería *assert* en realidad define una macro *assert*



```
#define assert ( test )
```

Esta macro se expande como un bloque **if**, en el que se comprueba la condición '**test**' y, dependiendo de si es o no verdadera, puede abortar el programa. Si '**test**' se evalúa como *cero* (es decir, si es falsa) entonces se aborta el programa y se imprime un mensaje de salida en el que se incluyen la condición '**test**', el nombre del archivo fuente y el número de línea en la que se llamó a `assert()`.

Vamos a ver esto en detalle tomando como referencia el primer ejemplo que vimos: vamos a modificar la función *main* para que utilice la llamada a *assert*:



```
1 #include <stdio.h>
2 #include <assert.h>
3
4 //Representamos temperaturas mediante números enteros
5 //farCel: int -> int
6 //El parámetro representa una temperatura en Fahrenheit y,
7 // se retorna su equivalente en Celsius.
8 // entrada: 32, salida: 0
9 // entrada: 212, salida: 100
10 // entrada: -40, salida: -40
11 int farCel(int f) {
12     return (f-32)*5/9;
13 }
14
15 int main() {
16     assert(farCel(32) == 0);
17     assert(farCel(212) == 100);
18     assert(farCel(-40) == -40);
19     return 0;
20 }
```

Como se puede ver, estamos verificando que el valor obtenido en la llamada a la función *farCel* coincide con lo esperado. Al cumplirse todos los casos, los test se pasan sin problemas (sin aviso también!). En cambio si, por ejemplo, hubieramos puesto en la línea 17:



```
assert(farCel(212) != 100);
```

La salida que obtendríamos seria:



```
1
2 File: prueba.c, Line 17
3
4 Expression: farCel(212) != 100
5
6 This application has requested the Runtime to terminate it in an ↵
   unusual way.
7 Please contact the application's support team for more ↵
   information.
```