

Funciones en C, pasaje de parámetros.

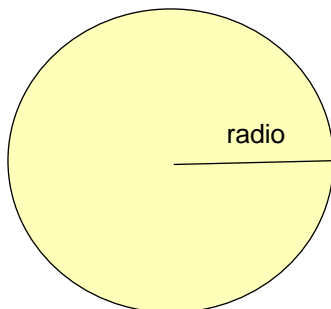
1

Estructura de un programa en lenguaje C

- Comandos para el preprocesador (CPP)
 - **#include**
 - **#define**
- Definición de **variables globales**.
- **Prototipos de funciones** (define el nombre de la función, el tipo que retorna y los tipos de los parámetros que recibe).
- **Funciones** (incluyendo la función main()).

2

Primer Ejemplo: Cálculo del área de un círculo



$$\text{area} = 3,14159 * \text{radio}^2$$

- Queremos construir un programa que calcule el área de un círculo.
- El programa debe recibir como entrada el radio del círculo.
- Al final, el programa debe imprimir por pantalla el área.

3

Usando solo la función main()

```
#include <stdio.h>
#define PI 3.14159
```

```
void main() {
    float a, radio;

    printf("Ingrese radio del círculo > ");
    scanf("%f", &radio);
    printf("\n");
    a = PI*radio*radio;
    printf("Area del círculo %f \n", a);
}
```

La
función
main()
realiza
varias
tareas

Leer datos
Calcular área
Imprimir resultados

Cuando
tenemos
programas
muy grandes,
no es
conveniente
colocar todas
las
instrucciones
en una sola
función

4

Definición de funciones

- El formato de definición de una función sería el siguiente:

```
tipoDato nombreFunción([listaParámetrosFormales]){  
  //Cuerpo de la función ...  
  [return expresiónDeRetorno]  
}
```

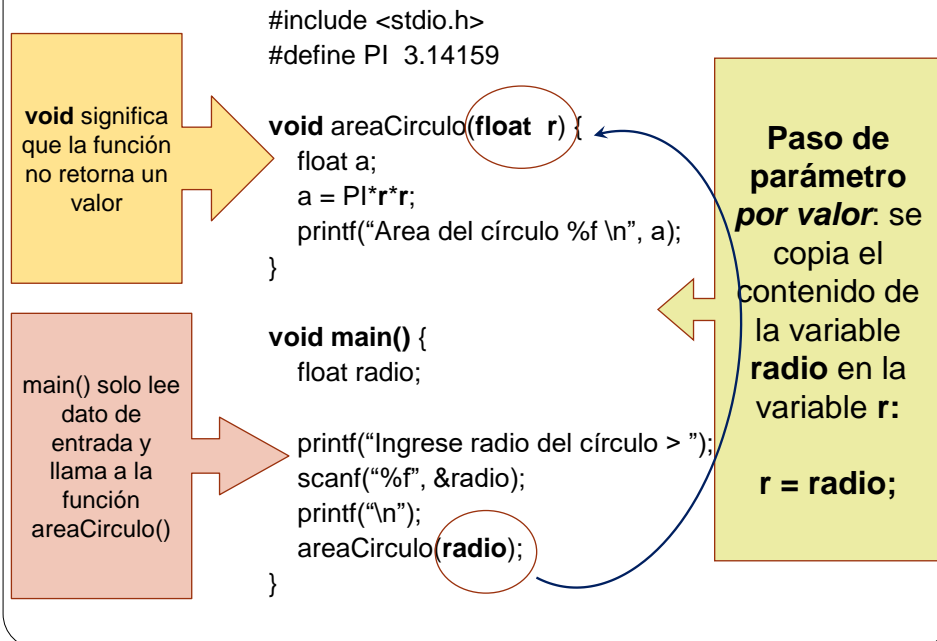
- Una función devuelve un dato, por lo que será necesario indicar el tipo de dato que devuelve. **tipoDato** podrá contener alguno de los tipos de datos utilizados para la declaración de variables

5

- En algunos casos la función no devolverá un valor, entonces se utiliza la palabra reservada **void** como tipo de la función para indicar que el valor de retorno de la función no está requerido.
- El **nombreFunción** será un identificador válido.
- Una función devolverá un valor a partir de los datos que se le pasan como argumentos. En el prototipo de función deberán constar el nombre de los argumentos y el tipo de cada uno de ellos en la **listaParámetrosFormales**.
- Si una función no tiene parámetros, en su llamada se incluirá un parámetro **void** para indicar que se trata de un dato no requerido.

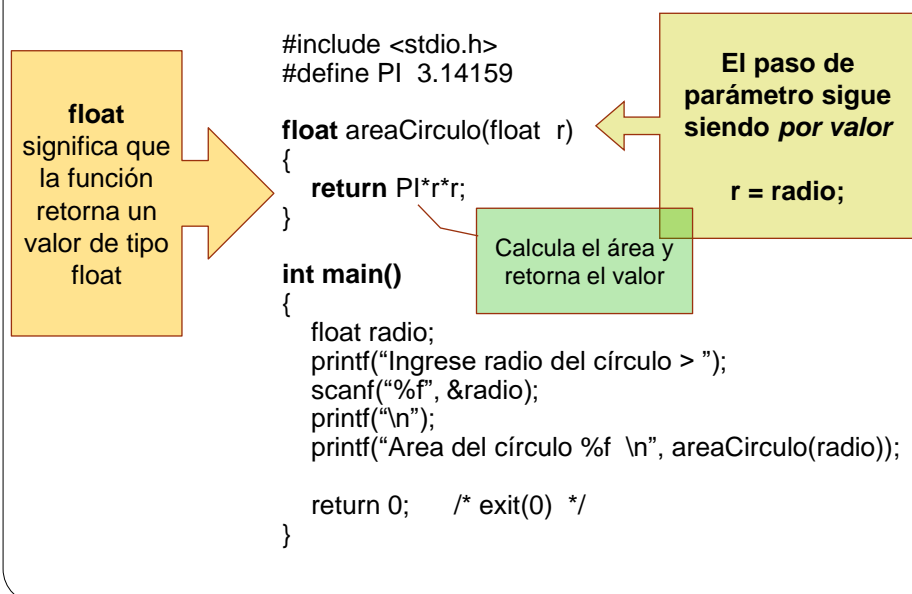
6

Usamos un programa con dos funciones



7

Función que retorna un valor



8

Usando prototipo para definir una función

```
#include <stdio.h>
#define PI 3.14159
```

```
float areaCirculo(float);
```

```
int main()
```

```
{
    float radio;
    printf("Ingrese radio del círculo > ");
    scanf("%f", &radio);
    printf("\n");
    printf("Area del círculo %f \n", areaCirculo(radio));

    return 0;    /* exit(0) */
}
```

```
float areaCirculo(float radio)
{
    return PI*radio*radio;
}
```

Cuando las funciones son escritas después de main(), se debe definir el prototipo de éstas al comienzo.

9

Implementación de funciones

Sin retornar un valor:

```
void areaCirculo(float radio)
{
    float a;
    a = PI*radio*radio;
    printf("Area del círculo %f \n", a);
}
```

Con retorno de un valor:

```
float areaCirculo(float radio)
{
    return PI*radio*radio;
}
```

10

Uso de la función

Usamos el valor que retorna una función:

- Como parámetro de otra función.
- Almacenamos el valor de retorno en una variable.
- El valor de retorno se usa directamente en una expresión, desde donde se llama a la función.

Si la función no retorna un valor, la llamada aparece aislada en una línea del programa.

11

Uso de la función: como parámetro de otra función

```
int main()
{
    float radio;

    printf("Ingrese radio del círculo > ");
    scanf("%f", &radio);
    printf("\n");

    printf("Area del círculo %f \n", areaCirculo(radio));

    return 0;    /* exit(0) */
}
```

12

Uso de la función: asignando valor de retorno a una variable

```
int main()
{
    float area, radio;

    printf("Ingrese radio del círculo > ");
    scanf("%f", &radio);
    printf("\n");

    area = areaCirculo(radio);

    printf("Area del círculo %f \n", area);

    return 0;    /* exit(0) */
}
```

13

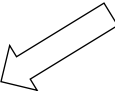
Uso de la función: El valor de retorno se usa dentro de una expresión

```
int main()
{
    float area, h, radio, volumen_cilindro;
    printf("Ingrese radio del círculo > ");
    scanf("%f", &radio);
    printf("\n");
    printf("Ingrese altura del cilindro > ");
    scanf("%f", &h);
    printf("\n");

    volumen_cilindro = h*areaCirculo(radio);

    return 0;    /* exit(0) */
}
```

Calcular el
volumen de
un cilindro



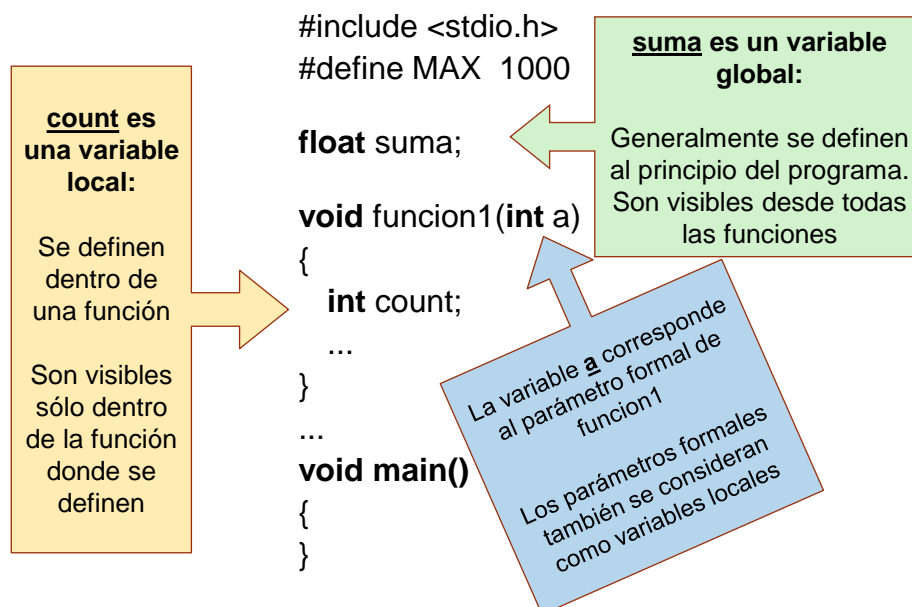
14

Ventajas de las funciones

- Permiten usar el **principio de dividir para conquistar**, en el sentido de que un problema grande y complejo lo podemos dividir en un conjunto de tareas más pequeñas y manejables.
- Construcción modular de un programa, que ayuda también a que éste sea más legible y comprensible.
- Permite evitar la redundancia de instrucciones.

15

Variables locales y variables globales



16

Ámbito o alcance de las variables

- El **ámbito o alcance de un identificador** es la parte del programa dónde se puede utilizar la variable.
- En C se puede distinguir tres tipo de ámbitos:
 - **Global:** Su ámbito es el programa dónde han sido definidas. Se trata de variables o funciones externas que se definen fuera de cualquier función y antes de su definición.
 - **Local.** Su ámbito es el bloque dónde han sido definidas. Se trata de aquellas variables definidas dentro de un bloque, es decir entre los símbolos { y }. Las variables declaradas dentro de una función, serán variables locales.
 - **Ámbito del prototipo.** Cuando se declaran los argumentos en un prototipo de función, su ámbito queda reducido a la declaración del propio prototipo.

17

Variables globales

- Una **variable global** se define fuera del cuerpo de cualquier función, generalmente al principio del programa, después de la definición de los archivos cabeceras y de la definición de constantes simbólicas y antes de cualquier función.
- El **ámbito de una variable global** son todas las funciones que componen el programa y cualquier función puede acceder a las variables globales como lectura y escritura.
- Su existencia perdura durante toda la ejecución del programa.
- Se debe minimizar el uso de este tipo de variables.

18

Variables locales

- Una **variable local** es aquella cuyo ámbito se restringe a la función que la ha declarado.
- Los parámetros formales, definidos en el encabezado de la función, se consideran **variables locales** a la función que las declara.
- No se puede hacer referencia a una variable local fuera de esa función.
- Existen mientras la función se está ejecutando, es decir, su almacenamiento es temporal.

19

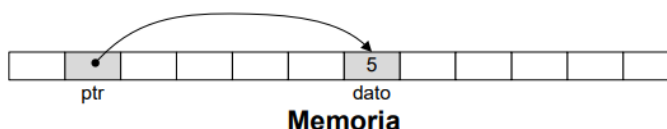
Variables locales y variables globales con el mismo nombre

- Cuando en un programa se definen variables locales y globales con el mismo nombre e incluso con el mismo tipo, entonces las variables locales y los parámetros formales son quienes tienen prioridad sobre las globales dentro de la función.
- Las variables locales y globales con el mismo nombre corresponden a localizaciones de memoria distintas, y obviamente con direcciones de memoria distintas.
- No puede haber dos variables locales con el mismo nombre dentro de la misma función, aunque sean de distinto tipo.
- No puede haber dos variables globales con el mismo nombre, aunque sean de distinto tipo.

20

Punteros

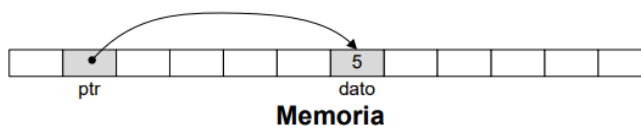
- Un puntero es una variable que contiene la dirección de memoria dónde está almacenado un objeto.
- Si ptr es un puntero, contendrá la dirección de memoria donde se almacena otro dato, por ejemplo, la variable dato.



21

Punteros y los operadores * y &

- Cuando se utilizan punteros, el **operador unario &** devuelve la dirección de memoria de un objeto, de forma que **ptr = &dato;** guardaría en la variable ptr la dirección de memoria de la variable dato.

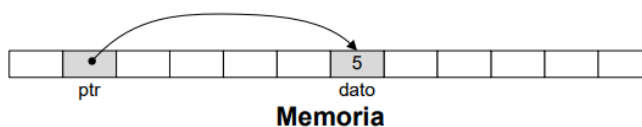


- El **operador de indirección o desreferencia** es el **operador unario *** y devuelve el contenido del objeto referenciado, es decir, el contenido de la dirección de memoria a la que apunta el puntero.

22

Punteros y los operadores * y &

- De esta forma, `printf("%i",*ptr);` devolvería el contenido de dato, es decir, un 5.



- La declaración de las variables `ptr` y `dato`, se haría de la siguiente forma:

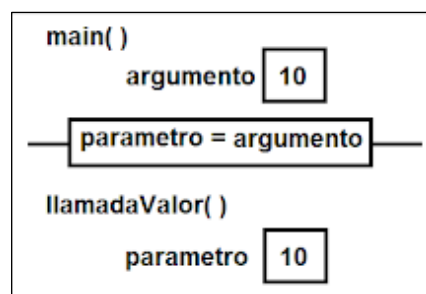
```
int dato = 5;
int *ptr;
```

- Con esto se está declarando a la variable `ptr` como un puntero capacitado para apuntar a una variable de tipo entero.

23

Pasaje de parámetros por valor

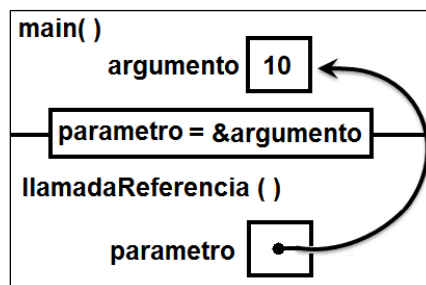
- Cuando se hace el **pasaje de argumentos por valor**, lo que se pasa es una **copia del valor del parámetro actual** y el parámetro formal (el de la función) se carga con esa copia.
- Al tratarse de una copia, cualquier modificación se realizará sobre la copia, sin alterar el parámetro actual.



24

Pasaje de argumentos por referencia

- Cuando se hace el **pasaje de argumentos por referencia**, lo que se pasa es la **referencia de una variable**, es decir, la dirección dónde está almacenada.



- De esta forma, el parámetro formal (el de la función) se carga con la dirección de la variable, por lo que **cualquier modificación que se haga sobre el argumento, se estará haciendo directamente sobre la dirección de memoria del parámetro actual modificando su valor.**

25

Paso de argumentos por referencia

- Como se pasa es la dirección de la variable (un puntero) los argumentos deberán de ser también punteros.
- Para acceder al contenido de la dirección de memoria habrá que utilizar el operador de indirección *.
- Por ejemplo, se quiere hacer una función que intercambie el contenido de dos variables enteras, como los parámetros formales tendrán que intercambiar su valor, entonces el paso de argumentos se debe hacerse por referencia

```
void intercambiarEnteros(int *x, int *y){
    int aux;
    aux = *x;
    *x = *y;
    *y = aux;
}
```

```
...
int a = 5;
int b = 8;
intercambiarEnteros(&a,&b);
...
```

26

```

#include <stdio.h>

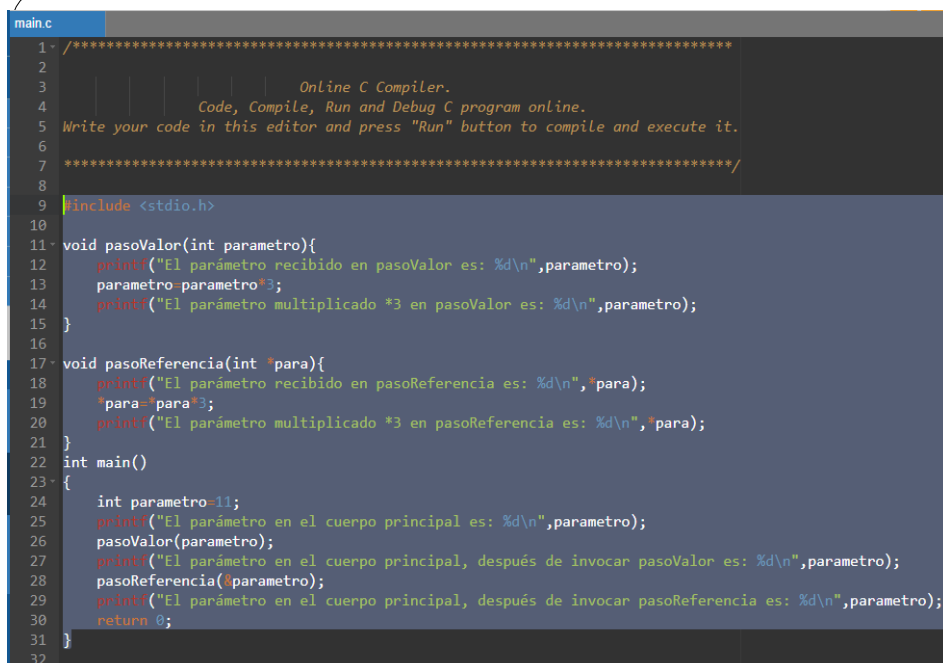
void pasoValor(int parametro){
    printf("El parámetro recibido en pasoValor es: %d\n",parametro);
    parametro=parametro*3;
    printf("El parámetro multiplicado *3 en pasoValor es: %d\n",parametro);
}

void pasoReferencia(int *para){
    printf("El parámetro recibido en pasoReferencia es: %d\n",*para);
    *para=*para*3;
    printf("El parámetro multiplicado *3 en pasoReferencia es: %d\n",*para);
}

int main()
{
    int parametro=11;
    printf("El parámetro en el cuerpo principal es: %d\n",parametro);
    pasoValor(parametro);
    printf("El parámetro en el cuerpo principal, después de invocar pasoValor es:
%d\n",parametro);
    pasoReferencia(&parametro);
    printf("El parámetro en el cuerpo principal, después de invocar pasoReferencia
es: %d\n",parametro);
    return 0;
}

```

27



```

main.c
1- /******
2-
3- Online C Compiler.
4- Code, Compile, Run and Debug C program online.
5- Write your code in this editor and press "Run" button to compile and execute it.
6-
7- *****/
8-
9- #include <stdio.h>
10-
11- void pasoValor(int parametro){
12-     printf("El parámetro recibido en pasoValor es: %d\n",parametro);
13-     parametro=parametro*3;
14-     printf("El parámetro multiplicado *3 en pasoValor es: %d\n",parametro);
15- }
16-
17- void pasoReferencia(int *para){
18-     printf("El parámetro recibido en pasoReferencia es: %d\n",*para);
19-     *para=*para*3;
20-     printf("El parámetro multiplicado *3 en pasoReferencia es: %d\n",*para);
21- }
22- int main()
23- {
24-     int parametro=11;
25-     printf("El parámetro en el cuerpo principal es: %d\n",parametro);
26-     pasoValor(parametro);
27-     printf("El parámetro en el cuerpo principal, después de invocar pasoValor es: %d\n",parametro);
28-     pasoReferencia(&parametro);
29-     printf("El parámetro en el cuerpo principal, después de invocar pasoReferencia es: %d\n",parametro);
30-     return 0;
31- }
32-

```

28

```
#include <stdio.h>

void pasoValor(int parametro){
    printf("El parámetro recibido en pasoValor es: %d\n",parametro);
    parametro=parametro*3;
    printf("El parámetro multiplicado *3 en pasoValor es: %d\n",parametro);
}

void pasoReferencia(int *para){
    printf("El parámetro recibido en pasoReferencia es: %d\n",*para);
    *para=*para*3;
    printf("El parámetro multiplicado *3 en pasoReferencia es: %d\n",*para);
}

int main()
{
    int parametro=11;
    printf("El parámetro en el cuerpo principal es: %d\n",parametro);
    pasoValor(parametro);
    printf("El parámetro en el cuerpo principal, después de invocar pasoValor es: %d\n",parametro);
    pasoReferencia(&parametro);
    printf("El parámetro en el cuerpo principal, después de invocar pasoReferencia es: %d\n",parametro);
    return 0;
}
```

aporte

```
El parámetro multiplicado *3 en pasoValor es: 33
El parámetro en el cuerpo principal, después de invocar pasoValor es: 11
El parámetro recibido en pasoReferencia es: 11
El parámetro multiplicado *3 en pasoReferencia es: 33
El parámetro en el cuerpo principal, después de invocar pasoReferencia es: 33
```