



UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Pase de Parámetros

Prof. Judith Barrios Alborno

Departamento de Computación
Escuela de Ingeniería de Sistemas
Facultad de Ingeniería
Universidad de Los Andes
Semestre A_2013

Parámetros Formales

- ▶ Los **PARAMETROS FORMALES** son **VARIABLES LOCALES**
 - La única diferencia es el lugar donde son declarados:
 - en la **definición del subprograma** (*lista_parámetros_formales*)



Cuando se pasa un valor desde un subprograma “**llamador o invocador**” a un subprograma “**llamado o invocado**”

- se crea una *variable temporal* dentro del subprograma invocado
- una vez que termina la ejecución del subprograma invocado y se retorna al programa invocador, dicha variable *deja de existir*

Pase de Parámetros

- ▶ Dos formas:
 - **Por valor**
 - Por referencia



Por valor.

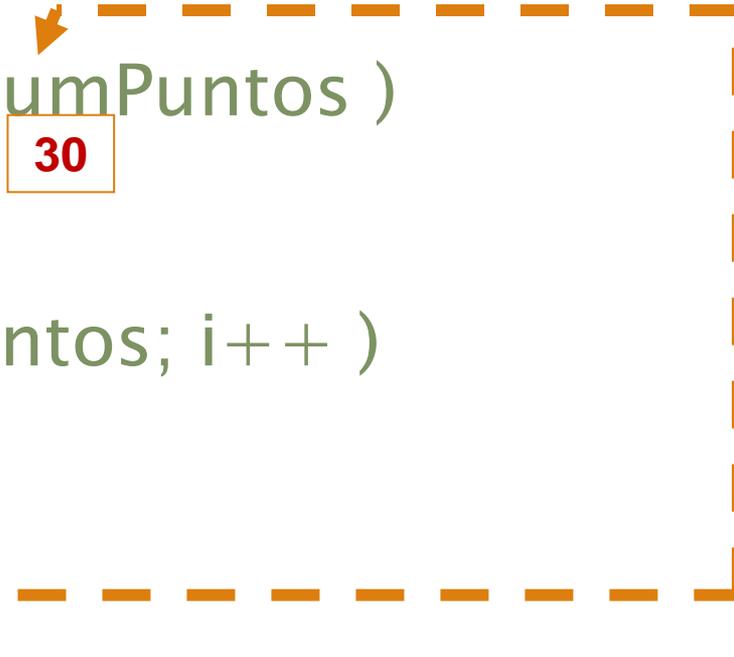
Los parámetros formales correspondientes **reciben una COPIA** de los valores (*constante, variable, resultado de la evaluación de una expresión*) de los parámetros actuales

- ▶ los cambios que se produzcan en ellos – **en los parámetros formales** – por efecto de la ejecución del subprograma, **NO afectan** el valor de los **parámetros actuales**

En **C** todas las llamadas, **por omisión**, son por valor

Ejemplo 1

```
void dibujarPuntos( int numPuntos )  
{  
    int i;  
    for ( i = 1; i <= numPuntos; i++ )  
        printf (".");  
}  
void main( )  
{  
    dibujarPuntos( 30);  
}
```

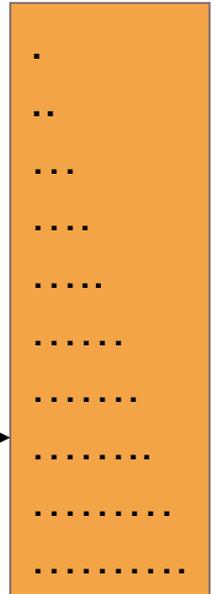


Ejemplo 2

Valor actual de **i**

```
void dibujarPuntos( int numPuntos )
{ int i;          /* i del procedimiento*/
  dibujarPuntos
  for ( i = 1; i <= numPuntos; i++ )
    printf (".");
}
void main( )
{ int i;          /* i del programa principal*/
  for ( i = 1; i <= 10; i++ )
  {  dibujarPuntos( i );
  }
}
```

Salida →





UNIVERSIDAD
DE LOS ANDES
MÉRIDA VENEZUELA

Ejemplo 3

```
#include <stdio.h>

void modificar(int);

void main()
{ int a = 2;
  printf (“antes de la llamada%d\n”, a);
  modificar(a);
  printf (“despues de la llamada %d\n” , a );
}

void modificar (int a)
{ a *= 3;
  printf (“desde la funcion %d\n”, a);
}
```

Pase de Parámetros

- ▶ Paso de parámetros
 - Por valor
 - *Por referencia o por dirección*



Por Referencia

- ▶ Se **pasa** a la función la dirección de memoria del parámetro actual
 - una **variable** pasada como parámetro actual es compartida
 - puede ser **accedida** y **modificada** durante la ejecución de la función



Apuntador o puntero

Apuntador: Variable especial cuyo contenido es la dirección o localización de memoria de otra variable.

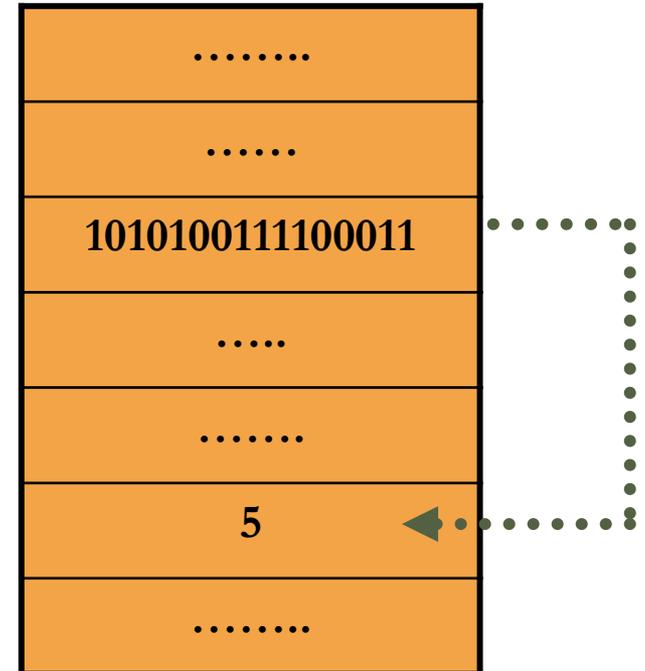
Ap1 es una variable de tipo apuntador

A una variable de tipo entero

```

.....
1101011001110110
Ap1 1100011100101111
1001000101001000
1001000100010001
A 1010100111100011
.....

```





Parámetros por valor vs. Parámetros por referencia

- ▶ El paso de parámetros **por referencia** permite **devolver** varios valores desde una función a través de los parámetros actuales
- ▶ Si una función **devuelve** un solo valor usando la sentencia *return*, todos sus parámetros deben ser pasados **por valor**



Parámetros por valor vs. Parámetros por referencia

- ▶ Si se pasa el valor de una variable (*parámetro actual*) a una función, ésta puede **usar el valor** de la variable **pero no lo** puede **cambiar**
- ▶ Si se pasa la dirección – por referencia– de una variable (*parámetro actual*) a un procedimiento, éste **puede cambiar el valor** de la variable



Pase de parámetros por referencia en C/C++

```
void nombreDelProcedimiento (tipo &parámetroFormal1, ...)  
{  
    /*Cuerpo del procedimiento*/  
}
```

Llamada al procedimiento

nombre *DelProcedimiento*(parámetroActual1, ...)

&: la dirección de

Ejemplo 4

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

```
void Cuadrado( int num, int &AptCuadrado )  
{  
    AptCuadrado = num * num;  
}
```

```
void main( )  
{
```

```
    int c = 12;  
    Cuadrado( 5, c);  
    printf ("El cuadrado de 5 es %d\n", c);  
}
```

Por valor

Por referencia

Ejemplo 5

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

```
/* Prototipo de la funcion f1 */
```

```
void f1(int, int);
```

```
/* Prototipo de la funcion f2 */
```

```
void f2 (int &, int &);
```

```
void main( )
```

```
{
```

```
int u = 1, v = 3;
```

```
printf (“u =%d v = %d antes de la llamada  
a f1” \n“, u, v);
```

```
f1(u, v);
```

```
printf (“u =%d v = %d despues de la  
llamada a f1” \n“, u, v);
```

```
f2(u, v);
```

```
printf (“u =%d v = %d despues de la  
llamada a f2” \n“, u, v);}
```

```
void f1 (int a, int b)
```

```
/* por valor*/
```

```
{
```

```
a = 0;
```

```
b = 0;
```

```
printf (“a =%d b = %d dentro de f1\n“, a , b);
```

```
return;
```

```
}
```

```
void f2 (int &Apta, int &Aptb) /* por */
```

```
/* referencia*/
```

```
{
```

```
Apta = 0;
```

```
Aptb = 0;
```

```
printf ( “a = %d b = %d dentro de f2\n“, Apta,  
Aptb);
```

```
return;
```

```
}
```

Ejemplo 6

```
#include <stdio.h>

int cuadradoPorValor (int);
void cuadradoPorReferencia (int &);

void main ()
{ int x = 2, z = 4;

printf (“x=%d antes de la llamada a
cuadradoPorValor\n“, x);
printf (“valor devuelto por cuadradoPorValor:
%d\n“, cuadradoPorValor(x);
printf (“x= %d despues de la llamada a
cuadradoPorVal \n“, x);
printf (“z= %d antes de la llamada a
cuadradoPorReferencia\n“, z);
```

```
printf (cuadradoPorReferencia(z));
printf (“z = %d despues de la llamada a
cuadradoPorReferencia\n“, z);
}

int cuadradoPorValor (int a)
{
return a *= a;
}

void cuadradoPorReferencia (int &cRef)
{
cRef *= cRef;
}
```

Ejercicio 1

Diseñe una función que reciba los valores enteros de x y $n \geq 0$ como parámetros de entrada y devuelva el valor de x^n como salida

<p style="text-align: center;">potencia(Entero x, Entero n): Entero <pre>{pre: n ≥ 0} {pos: pot ≥ 0 }</pre></p>		
<p>1 pot = 1 2 Repita para j = 1, j ≤ n, j = j + 1 pot = pot * x frp 3 regrese pot</p>	<ul style="list-style-type: none"> ● pot: Entero. Acumulador para x^n ● j: Entero. Contador del repita para 	
<p>1 x = 1, n = 0 => pot = 1 2 x = 2, n = 4 => pot = 16 3 x = -9, n = 2 => pot = 81</p>		<p>Caso exitoso Caso exitoso Caso exitoso</p>

Ejercicio 2

Diseñar un programa que tome un conjunto de pares de números enteros a y b , y calcule la potencia a^b de cada par. El fin de entrada de datos viene dado por $a = -1$ y $b = -1$

Análisis E-P-S

Entrada: Valores de a y $b \in \mathbb{Z}$

Proceso: repetidamente calcular a^b mediante la función entera $\text{potencia}()$, hasta que los valores de a y b sean -1

Salida: para cada iteración imprimir el valor resultante si $b \geq 0$, de lo contrario indicar el error del dato b



Algoritmo ejercicio 2

variasPotencias()

{pre: }

{pos: }

1	ban = Cierto	<ul style="list-style-type: none">● resulta: Entero. Valor de la potencia de a a la b.● a, b: Entero. Base y exponente, respectivamente● band: Lógico. Bandera para terminar el programa● potencia(Entero, Entero): Entero Función para calcular la potencia de a a la b
2	hacer escribir "Introduzca los valores a y b" leer a, b Si($a = -1 \wedge b = -1$) entonces band = Falso sino Si ($b \geq 0$) entonces resulta = potencia(a, b) escribir a, " a la ", b, " es ", resulta sino escribir "Error, b debe ser ≥ 0 " fsi fsi mientras(ban = Cierto)	
1	$a = 1, b = 0 \Rightarrow resul = 1$	Caso exitoso
2	$a = 2, b = 4 \Rightarrow resul = 16$	Caso exitoso
3	$a = -1, b = -1 \Rightarrow ban = Falso$	Fin del programa

Ejercicio 3

Diseñar una función SumaNegativos () que permita sumar los elementos negativos de una lista de n números reales

sumaNegativos(): Real		{pre: }	{pos: resultado \leq 0 }
1 Escribir “¿Cuántos números son?” 2 Leer n 3 resultado = 0 4 Repita para j = 1, n, 1 Escribir “Introduzca un valor real” Leer x Si(x < 0) entonces resultado = resultado + x fsi frp 5 regrese resultado	<ul style="list-style-type: none"> ● n: Entero. Número total de valores reales que serán suministrados por teclado ● j: Entero. Contador del repita para ● resultado: Real. Acumulador para la suma de los valores negativos ● x: Real. Valor real suministrado 		
1 n = 0 => resultado = 0 2 x = 2, -1, 4, 3, n = 4 => resultado = -1 3 x = -9, -1, n = 2 => resultado = -10	Caso exitoso Caso exitoso Caso exitoso		

Ejercicios

1. Escribir una función que compruebe si un número dado es correcto, en cuyo caso la función deberá devolver el valor q . En caso negativo, la función devolverá el valor -1

Considere los siguientes casos para decidir si un número es o no correcto:

- El número es correcto si se encuentra en el rango definido por dos valores constantes MINIMO y MAXIMO.
- El número es correcto si es uno de tres valores constantes VALOR1, VALOR2 o VALOR3.
- El número es correcto si se encuentra en el rango definido por dos valores constantes MINIMO y MAXIMO, o bien es igual al valor constante VALOR1.
- El número es correcto si se encuentra en el rango definido por dos valores constantes MINIMO1 y MAXIMO1, o bien en el definido por los valores constantes MINIMO2 y MAXIMO2

Ejercicios

2. Diseñar una función que tome un carácter como parámetro de entrada y devuelva un número que indica el tipo de carácter introducido. Este podrá ser uno de los siguientes tipos: (1) letra mayúscula de la 'A' a la 'Z', (2) letra minúscula de la 'a' a la 'z', (3) dígito del '0' al '9'.
3. Diseñar una función que tome como parámetros de entrada dos instantes de tiempo expresados en horas, minutos y segundos e indique si el primero es anterior al segundo.
4. Diseñar las funciones que sean necesarias para leer cuatro números reales, calcular y escribir su producto, suma y su media aritmética.

Ejercicios

5. Dada la longitud, ancho y profundidad (en pies) de una piscina, el volumen se calcula según la siguiente fórmula:

$$\text{volumen} = \text{longitud} * \text{ancho} * \text{profundidad}$$

Dado que un pie cúbico de agua es equivalente a 7.8 galones, la capacidad de agua de la piscina viene dada por la fórmula:

$$\text{capacidad} = \text{volumen} * 7.8$$

Si la rata de flujo de agua en la piscina es de 20 galones por minuto, entonces el tiempo (en horas) que se requiere para llenar la piscina se calcula mediante la fórmula:

$$\text{Tiempo} = \text{capacidad} / 20 / 60$$

