Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 5

Modularización, Abstracción

Acciones

Noticias

2 de abril: Día del Veterano y de los Caídos en la Guerra de Malvinas – 36 años (1982-2018)

Con orgullo y sin olvidar a los gauchos, con Rivero a la cabeza, y los pibes del '82 que defendieron las islas de los imperialistas. Sin olvidar a los genocidas y torturadores del 76 al 83 que, acostumbrados a matar gente indefensa, hablaban con valentía frente a la cámaras de televisión, pero no fueron al frente de batalla sino que mandaron a nuestros pibes.

Dice Eduardo Galeano: "*la Guerra de las Malvinas, guerra patria que por un rato unió a los argentinos pisadores y a los argentinos pisados, culmina con la victoria del ejército colonialista de Gran Bretaña. No se han hecho ni un tajito los generales y coroneles argentinos que habían prometido derramar hasta la última gota de sangre. Quienes declararon la guerra no estuvieron en ella ni de visita. Para que la bandera argentina flameara en estos hielos, causa justa en manos injustas, los altos mandos enviaron al matadero a los muchachitos enganchados por el servicio militar obligatorio, que más murieron de frío que de bala. No les tiembla el pulso: con mano segura firman la rendición los violadores de mujeres atadas, los verdugos de obreros desarmados.*"

Noticias

2 de abril: Día del Veterano y de los Caídos en la Guerra de Malvinas – 36 años (1982-2018)

Acto en la UNRC reconocimiento a los combatientes de Malvinas. En el acto se puso el nombre de COMBATIENTES DE MALVINAS al Anfiteatro 1 Pabellón 4, una d ella saulas más importantes de la Universidad.

Participaron docentes de los talleres de la Secretaría de Extensión y Desarrollo, Mario Gallo y Carolina Marconi, y parte del alumnado del taller de Mosaiquismo (Claudia Gigena, Silvia Garro, Maria Cristina Zapata y Patricia Dubini).

Noticias

El Gobierno recortó el presupuesto universitario

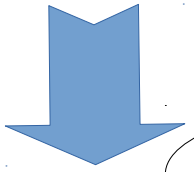
El Gobierno Nacional recortó $3.000.000.000 del presupuesto aprobado por el Congreso a las Universidades Nacionales. Un 3% menos para la UNRC, casi $45.000.000. Ello significa menos inversión y trabajo en la Ciudad, y la paralización del nuevo Pabellón de aulas licitado el año pasado.

La Nación frenó hasta el 2019 las nuevas obras en la UNRC

Estaba previsto que se iniciara una batería de aulas que había sido licitada a mediados del año pasado. Desde la conducción de la Universidad esperaban los fondos para iniciar los trabajos pero ahora desde Buenos Aires les comunicaron que no les librarán más recursos durante 2018

LOCALES Página 12

Modularización
Abstracción



Funciones **Acciones**

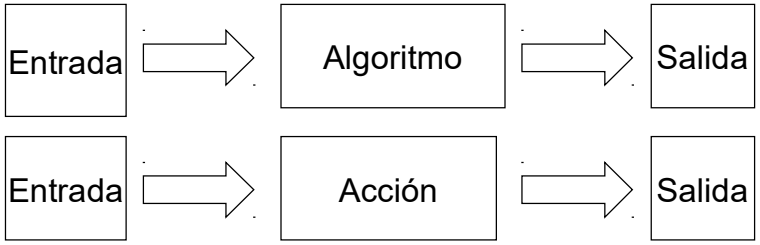
Bibliotecas

Modularización

- Las acciones son otra forma de ayudar a resolver problemas complejos del mundo real.
- Las acciones posibilitan construir módulos, para obtener soluciones independientes, y así aprovechar todas las ventajas de la modularización: productividad, reusabilidad, mantenimiento correctivo, facilidad de crecimiento, y mejor legibilidad.

Acciones como Algoritmos

- Las acciones son mini-algoritmos.
- Un algoritmo puede ser entendido como un procesador de información, con datos de entrada y de salida.
- Las acciones son similares a los algoritmos.
- Una acción puede transformar un estado en otro distinto.



Acciones con parámetros

- Las acciones son más efectivas cuando son módulos **autocontenidos**.
- Cuando un problema es muy complicado los programas escritos para resolverlo serán a su vez también complejos.
- Entonces, para poder encontrar una buena solución al problema recurrimos al diseño descendente y dividimos el problema en sub-problemas.

Acciones con parámetros

- Si las acciones que dan solución a los subproblemas son módulos **autocontenidos**, uno puede resolver y testear cada acción **independientemente** de las demás acciones.

- Para que las acciones sean **autocontenidas, no deben** hacer **referencia a declaraciones** (variables, constantes, tipos) que estén **fuera de dicha acción**, como por ejemplo del léxico del algoritmo principal.

Acciones con parámetros

- Para lograr que una acción pueda ser considerada un **algoritmo autocontenido**, la información debe poder ser transferida entre las acciones y el resto del algoritmo principal a través de lo que llamaremos ***parámetros***.

- Los parámetros permiten que una acción pueda manipular diferentes valores y por lo tanto la misma acción puede ser usada tantas veces como sea necesario en un mismo algoritmo.

Acciones
Estructura

- Se componen de:

1. Un encabezamiento: el cual tiene la palabra reservada **Acción** seguida de un identificador.

   Este identificador puede o no ser seguido de parámetros.

2. Declaraciones locales: esto es un léxico local donde se declaran las variables locales a la acción.
3. Bloque o Cuerpo de acciones ejecutables: encerradas entre **Inicio** y **Facción** se desarrollan el conjunto de acciones o composiciones (secuenciales, condicionales, etc.) que resuelven la especificación de la acción.

Acciones
Estructura

1. *Cabecera* — **Acción <identificador> (<lista de parámetros>)**
2. *Declaraciones* — **Léxico local** (si es necesario)

```
variables, constantes, acciones y
funciones propias de la acción
```

3. *Sentencias ejecutables*

```
Inicio
   <acción más simple>
    ...
   <acción más simple>
Facción
```

4. *Ubicación*

Las declaraciones de acciones se hacen en el léxico del algoritmo principal, después de las declaraciones de los identificadores del mismo (variables, etc.).

Acciones
Invocación y ejecución

- Una acción se ejecuta indicando su nombre y los parámetros. Esto se conoce como “llamado” o “invocación” de la acción. El resultado es la ejecución de las acciones contenidas en esa acción.
- Luego de ejecutar una acción el algoritmo continúa ejecutando las acciones siguientes a la llamada o invocación.

Acciones con parámetros
Motivación
Ejemplo: Intercambio de variables

Supongamos que deseamos resolver el siguiente problema:

> En una carrera de 100 metros compiten tres corredores. Al llegar a la meta se toman los tiempos de cada uno, en segundos. Los tiempos son distintos entre sí, es decir, no hay empate. Luego, la mesa de control informa los tiempos, ordenados de menor a mayor. ¿Puedes colaborar con la mesa de control ordenando los tiempos que recibe para informarlos en orden creciente?

Pensemos una solución.....

Un algoritmo que solucione el problema planteado puede ser:

Intercambio de variables

Análisis

Etapa 2 (identificar los datos de entrada, los resultados y las relaciones o subproblemas):

- **Dato**: **a**, **b** y **c** son los tiempos de cada corredor. Son números.
- **Resultado**: **p**, **s** y **t** son los tiempos ordenados de menor a mayor. Son números.
- **Relaciones o subproblemas**: los tiempos **a**, **b** y **c**, son distintos entre sí. **p** es el menor de a, b y c. **s** es valor intermedio entre a, b y c. **t** es el mayor de los tiempos a, b y c.

Intercambio de variables

Diseño

```
Acción clasificarTresValores
Léxico local
  a, b, c, aux, p, s, t ∈ Z
Inicio
  Entrada:a b c
  p ← a
  s ← b
  t ← c
  según
    p<s y p<t: nada
    s<p y s<t: aux ← s
               s ← p
               p ← aux
    t<p y t<s: aux ← t
               t ← p
               p ← aux
  fsegun
  según
       s<t: nada
       t<s: aux ← s
            s ← t
            t ← aux
  fsegun
  Salida:p s t
Facción
```

Para cuando necesitamos usar variables que solo participan dentro de ésta acción

Intercambio de variables

Reescribimos el algoritmo anterior de la siguiente forma:

```
Acción clasificarTresValores
Léxico local
  a, b, c, aux, p, s, t ∈ Z
Inicio
  Entrada:a b c
  p ← a
  s ← b
  t ← c
  según
    p<s y p<t: nada
    s<p y s<t: intercambia-ps
    t<p y t<s: intercambia-pt
  fsegún
  según
    s<t: nada
    t<s: intercambia-st
  fsegún
  Salida:p s t
Facción
```

```
aux ← s
s ← p
p ← aux
```

```
aux ← t
t ← p
p ← aux
```

```
aux ← s
s ← t
t ← aux
```


Intercambio de variables

```
Acción intercambia-ps
Léxico local
  aux ∈ Z
Inicio

  aux ← p
  p ← s
  s ← aux

Facción
```

```
Acción intercambia-pt
Léxico local
  aux ∈ Z
Inicio

  aux ← p
  p ← t
  t ← aux

Facción
```

```
Acción intercambia-st
Léxico local
 aux ∈ Z
Inicio

  aux ← s
  s ← t
  t ← aux

Facción
```

¿Qué cosas tienen en común éstas 3 acciones?

Intercambio de variables

Todas tienen el mismo formato:

```
Acción Intercambiar-♣♦
Léxico local
  aux ∈ Z
Inicio

  aux ← ♣
  ♣ ← ♦
  ♦ ← aux

Facción
```


Intercambio de variables

Podemos definir una única acción utilizando *parámetros*:

```
Acción Intercambiar(x, y ∈ Z)
Léxico local
  aux ∈ Z
Inicio

  aux ← x
  x ← y
  y ← aux

Facción
```

Aún están incompletos los parámetros. Falta el tipo de pasaje de parámetro.

El algoritmo quedaría: *(ver siguiente diapositiva)*

Intercambio de variables

Entonces clasificarTresValores quedaría:

```
Acción clasificarTresValores
Léxico local
  a, b, c, aux, p, s, t ∈ Z
  Acción Intercambiar(x, y ∈ Z)
Inicio
  Entrada:a b c
  p ← a
  s ← b
  t ← c
  según
    p<s ∧ p<t: nada
    s<p ∧ s<t: Intercambiar(p,s)
    t<p ∧ t<s: Intercambiar(p,t)
  fsegún
  según
    s<t: nada
    t<s: Intercambiar(s,t)
  fsegún
  Salida: p,s,t
Facción
```

```
Acción Intercambiar(x, y ∈ Z)
Léxico local
  aux ∈ Z
Inicio
  aux ← x
  x ← y
  y ← aux
Facción
```


Acciones con parámetros
Ejemplo

Se tiene que pintar de color amarillo una señal de peligro contenida en un cartel para la vía pública. Se necesita saber cuánto es el área a pintar.

- **Análisis**

Datos: base y altura de la señal de peligro.
Son números. Nombres: **base** y **altura.**
Resultado: área a pintar. Es un número. Nombre: **area**
Relaciones o subproblemas:
- Obtener la base y altura (base y altura)
- Calcular el área del triángulo (area): area = (base * altura) /2
- Informar el área (area)

Acciones con parámetros
Ejemplo

- **Diseño**

Los datos de entrada pueden ser parámetros de entrada de la acción, y el resultado puede ser parámetro de salida de la acción.

Parámetros de entrada: base y altura (actuales). baseTri y alturaTri (formales)
Parámetro de salida: area (actual). areaTri (formal)

```
Acción CalcularAreaTriángulo(baseTri, alturaTri, areaTri ∈ R)
Inicio
  areaTri ← (baseTri*alturaTri)/2
Facción
```

Aún están incompletos los parámetros. Falta el tipo de pasaje de parámetro.

Acciones con parámetros
Ejemplo

- **Diseño**

Parámetros de entrada: base y altura (actuales). x e y (formales).

```
Acción ObtenerDatos(x ∈ R, y ∈ R)
Inicio
  Entrada:x y
Facción
```

Parámetro de entrada: area (actual). a (formal)

```
Acción InformarResultados(a ∈ R)
Inicio
  Salida:a
Facción
```

Aún están incompletos los parámetros. Falta el tipo de pasaje de parámetro.

Acciones con parámetros
Invocación o llamado - Ejemplo

```
Algoritmo CalcularAreaTriángulo
Léxico
  base, altura ∈ R     //base y altura del triángulo
  area ∈ R             //área del triángulo
  Acción ObtenerDatos(x ∈ R, y ∈ R)
  Inicio
    Entrada:x y
  Facción
  Acción CalcularAreaTriángulo(baseTri ∈R,alturaTri ∈R,areaTri ∈R)
  Inicio
    areaTri ← (baseTri*alturaTri)/2
  Facción
  Acción InformarResultados(a ∈ R)
  Inicio
    Salida:a
  Facción
Inicio
  ObtenerDatos(base, altura)
  CalcularAreaTriángulo(base, altura, area)
  InformarResultados(area)
Fin
```

Aún están incompletos los parámetros. Falta el tipo de pasaje de parámetro.

Acciones con parámetros
Invocación o llamado

- **Observar que el tipo de las variables pasadas en la invocación son correspondientes con los tipos de las variables que se ponen en al definición de la acción.**

```
Léxico
  ...
  base, altura ∈ R
  Acción ObtenerDatos(x ∈ R, y ∈ R)
  ...
Inicio
 ...
  ObtenerDatos(base, altura)
 ...
Fin
```


Acciones con parámetros
Invocación o llamado

La invocación a una acción con parámetros se realiza de la misma manera que hemos visto que se invoca a una acción sin parámetros, agregando, a continuación del nombre de la acción, la lista de valores o variables a utilizar (parámetros actuales).

No puede ir un valor aquí.

- **Invocación con valores:**

```
CalcularAreaTriángulo(2.9, 6.4, area)
```

- **Invocación con variables:**

```
CalcularAreaTriángulo(base, altura, area)
```

- **Invocación mixta (con valores y variables):**

```
CalcularAreaTriángulo(4.5, altura, area)
```


Tipos de parámetros

Parámetros Formales

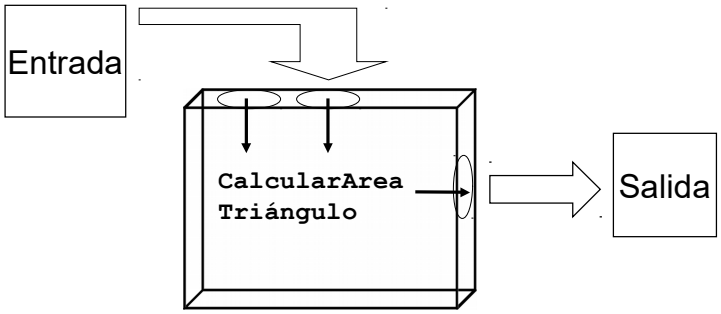
Nombre asignado en la cabecera de la acción a los objetos que serán manipulados en la acción.

Parámetros Actuales

Objetos que son pasados como datos en la invocación (o llamada) de una acción.
También llamados efectivos o reales.

Tipos de parámetros

- En la acción del triángulo, por ejemplo, los parámetros **x** e **y** no tienen valor asignado hasta que la acción **ObtenerDatos** es invocada. Recién en ese momento se le pasará el valor real de la **base** al parámetro **x,** y **altura** al parámetros **y**.



Acciones con parámetros
Invocación o llamado - Ejemplo

```
Algoritmo CalcularAreaTriángulo
Léxico
  base, altura ∈ R     //base y altura del triángulo
  area ∈ R             //área del triángulo
  Acción ObtenerDatos(x ∈ R, y ∈ R)
  Inicio
    Entrada:x y
  Facción
  Acción CalcularAreaTriángulo(baseTri ∈R,alturaTri ∈R,areaTri ∈R)
  Inicio
    areaTri ← (baseTri*alturaTri)/2
  Facción
  Acción InformarResultados(a ∈ R)
  Inicio
    Salida:a
  Facción
Inicio
   ObtenerDatos(base, altura)
   CalcularAreaTriángulo(base, altura, area)
   InformarResultados(area)
Fin
```

Parámetros Formales
x, y, baseTri, alturaTri, areaTri, a

Parámetros Actuales
base, altura, area

¿Cómo hacemos para distinguir cuáles son los parámetros de entrada y cuáles los de salida?

Tipos de pasaje de parámetros

Por Copia

vincula el parámetro formal a una variable local que contiene una copia del argumento.

Por Referencia

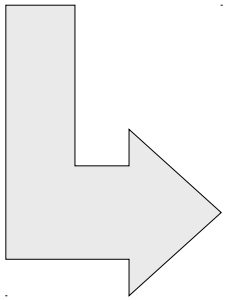
vincula el parámetro formal directamente al argumento en sí.

Tipos de pasaje de parámetros
Por COPIA

- **Entrada: Por Valor**
- **Salida: Por Resultado**
- **Entrada/Salida: Por Valor/Resultado**

Por ejemplo: C, C++, JAVA soportan solo el pasaje de parámetro por valor. ADA soporta los tres.

Tipos de pasaje de parámetros
Por Referencia

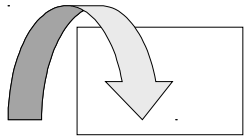


- **Es un mecanismo de pasaje de parámetros permite que el parámetro formal sea ligado directamente al argumento mismo (parámetro actual).**
- **Aparecen bajo varios "disfraces", o simulados, en algunos lenguajes de programación.**

Por ejemplo: C no admite mecanismos de parámetros de referencia directamente, pero podemos lograr el mismo "efecto" usando punteros (*).
ADA admite parámetros por referencia (constantes y variables). Pascal también, pero solo variables.

Tipos de pasaje de parámetros
Pasaje por Valor

Cláusula: dato



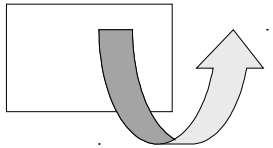
Los parámetros que poseen este tipo de pasaje de parámetro se lo conoce como ***parámetros de entrada***.

Cuando un parámetro es pasado por valor, el valor del parámetro actual es utilizado para inicializar el valor del parámetro formal.

Al asociarse el parámetro formal (puede ser una variable o una constante) sólo al valor inicial del parámetro actual, las modificaciones en el parámetro formal no afectan al parámetro actual.

Tipos de pasaje de parámetros
Pasaje por Resultado

Cláusula: resultado



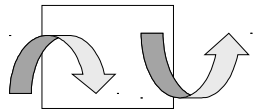
Los parámetros que poseen este tipo de pasaje de parámetro se lo conoce como ***parámetros de salida***.

Cuando un parámetro es pasado por resultado, no se transmite ningún valor durante la invocación. El valor inicial del parámetro formal es *indeterminado.*

Cuando el módulo finaliza su ejecución, el valor final del parámetro formal se asocia al parámetro actual, es decir se le asigna un resultado a la variable utilizada durante la invocación.

Tipos de pasaje de parámetros
Pasaje por Valor / Resultado

Cláusula: dato-resultado



Los parámetros que poseen este tipo de pasaje de parámetro se lo conoce como ***parámetros de entrada/salida***.

Es una combinación del pasaje por valor y por resultado.

El valor del parámetro actual es utilizado para inicializar el parámetro formal.

Cuando el módulo finaliza su ejecución, el valor final del parámetro formal se asocia al parámetro actual, es decir se actualiza el valor del parámetro actual.

Acciones con parámetros
Tipos de pasaje de Parámetros - Ejemplo

```
Algoritmo CalcularAreaTriángulo
Léxico
  base, altura ∈ R     //base y altura del triángulo
  area ∈ R             //área del triángulo
  Acción ObtenerDatos(resultado x ∈ R, y ∈ R)
  Inicio
    Entrada:x y
  Facción
  Acción CalcularAreaTriángulo(dato baseTri ∈ R, alturaTri ∈ R,
                               resultado areaTri ∈ R)
  Inicio
    areaTri ← (baseTri*alturaTri)/2
  Facción
  Acción InformarResultados(dato a ∈ R)
  Inicio
    Salida:a
  Facción
Inicio
  ObtenerDatos(base, altura)
  CalcularAreaTriángulo(base, altura, area)
  InformarResultados(area)
Fin
```

Aunque siendo **dato-resultado** funciona bien en este caso porque no se modifica la variable **a** dentro de la acción, **es muy peligroso para modificaciones futuras.**

No debe hacerse!!!

No hace falta que el nombre de los parámetros formales sea distinto que el de los actuales. Pero es aconsejable hacerlo hasta tanto se adquiera más práctica. Esto tiene sus pro y contras. ¿cuáles?

Acciones con parámetros
Estados: inicial y final (e_0, e_f)

¿Qué valores tienen las variables utilizadas en la invocación de una acción?
Para analizar el comportamiento de las *variables* utilizadas como *parámetros actuales* (según el tipo de pasaje de parámetros), usamos el concepto de ***estado***.
Los estamos pueden ser: ***eo*** (estado inicial), ***ef*** (estado final), ***ei*** (estado intermedio).
eo representa el valor que poseen las variables al inicio de un bloque, y ***ef*** al final.
El valor inicial de una variable se representa con su nombre en mayúscula y el subíndice 0 (cero). Por ejemplo, el valor inicial de **x** se escribe: **$x=X_0$**
Si la variable no está inicializada se coloca como indeterminado: **x=indet**

```
Acción ProSum(dato x,z ∈ R, dato-resultado w ∈ R, resultado prom ∈ R)
Léxico local
  a ∈ R
Inicio
  x ← x+1
  a ← w*2
  w ← x+z+w+a
  z ← z / 2
  prom ← w / 4
Facción
```

```
Algoritmo Ejemplo
Léxico
  i, j, k ∈ R
  Acción ProSum(dato x,z ∈ R, dato-resultado w ∈ R, resultado prom ∈ R)
Inicio
  Entrada: i j
  {e_o: i=I_0, j=J_0, k=indet}
  ProSum(i,1,j,k)
  {e_f: i=…., j=…..……., k=……..……..}
  Salida:i j k
Fin
```


Acciones con parámetros
Tipos de Parámetros

¿Qué valores tienen las variables utilizadas en `Prosum` luego de la invocación de dicha acción?

```
Acción ProSum(dato x,z ∈ R, dato-resultado w ∈ R, resultado prom ∈ R)
Léxico local
  a ∈ R
Inicio
  x ← x+1
  a ← w*2
  w ← x+z+w+a
  z ← z / 2
  prom ← w / 4
Facción
```

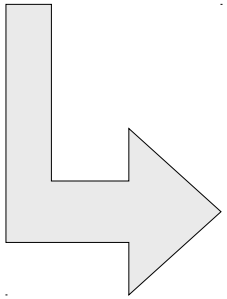
```
Algoritmo Ejemplo1
Léxico
  bb, cc, dd, pp ∈ R
  Acción ProSum...
Inicio
  dd ← 80
  cc ← 3
  bb ← 0
  {e_o: dd=80, cc=3, bb=0, pp=indet}
  ProSum(dd,cc,bb,pp)
  {e_f: dd=..., cc=..., bb=…, pp=…..}
  Salida:dd cc bb pp
Fin
```

```
Algoritmo Ejemplo2
Léxico
  f, g, h ∈ R
  Acción ProSum...
Inicio
  Entrada:f g
  {e_o: f=F_0, g=G_0, h=indet}
  ProSum(1,f,g,h)
  {e_f: f=..., g=…., h=...}
  Salida:f g h
Fin
```


Acciones con parámetros
Estructura general

- Las acciones vistas se pueden traducir a **C**. Pero **C** no tiene ***acciones puras***, sino que se usa el concepto de **`function`**.
- En **C** los tipos de pasaje de parámetros son solamente uno: ***por valor***.
- **C no** tiene pasaje de parámetro ***por resultado*** **ni por *valor/resultado***. Por lo tanto analizaremos como *traducir o simular* los tres tipos de pasaje de parámetros vistos anteriormente.

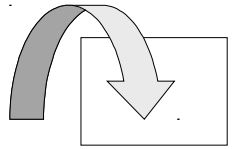
Tipos de pasaje de parámetros en C



- **Por Valor**

Tipos de pasaje de parámetros en C

Pasaje por Valor

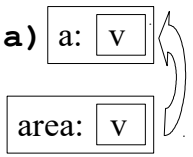


Cláusula:

El pasaje de parámetros por **valor** se ***traduce a C*** definiendo el parámetro formal solamente con su nombre y tipo, sin ninguna otra consideración adicional, es decir con la cláusula vacía (no se escribe nada) seguida del tipo del parámetro y el nombre. Como las acciones no devuelven un valor, pero las **`funtion`** de C si, se les coloca **`void (nulo)`** como tipo de retorno. También se dice que es una **`funtion`** sin tipo de retorno.

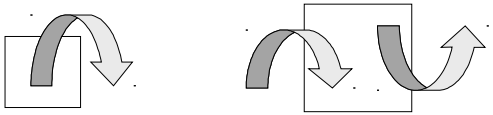
Ejemplo: **`void InformarResultados(float a)`**

Invocación: **`InformarResultados(area);`**



Tipos de pasaje de parámetros en C

Pasaje por Resultado y Valor/Resultado

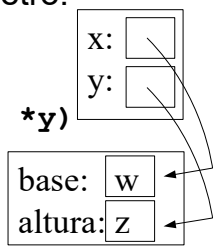


Cláusula: **`*`**

El eso del concepto de ***puntero*** (una variable que contiene la dirección de memoria de otra variable) permite ***simular en C*** los tipos de pasaje de parámetro por ***resultado*** y por ***valor/resultado***. Para ello se define el parámetro formal con el tipo y la cláusula **`*`** seguida del nombre del parámetro.

Ejemplo: **`void ObtenerDatos(float *x, float *y)`**

Invocación: **`ObtenerDatos(&base, &altura);`**



Tipos de pasaje de parámetros en C

Pasaje por Resultado y Valor/Resultado

¿Qué significa la cláusula *?

El **`*`** (asterisco), en C, significa ***puntero a algo...***

Un ***puntero*** es una variable que contiene la dirección de memoria de otra variable.

- ***El * es un operador*** (no significa producto). Se usa para declarar una variable de tipo puntero (**`float *areaTri`**).
- ***El & también es un operador.*** Se usa para obtener la dirección de memoria de otra variable y para asignar un valor a un puntero. El operador & antes de una variable retorna un puntero a esa variable del tipo de la misma (**`&area`**)

Tipos de pasaje de parámetros en C
Pasaje por Resultado y Valor/Resultado

```
#include <stdio.h>
int base; // 2 bytes
float prom; //4 bytes
char op1, op2; //1 byte c/u
int *pBase; // 4 bytes (en 32bits)
int main(){
  base= 5;
  pBase = &base;
  printf("Valor de base es: %d \n", base);
  printf("Direccion de base es: %p \n", &base);
  printf("Valor del puntero pBase es: %p \n", pBase);
  printf("Valor que apunta el puntero pBase es: %d \n", *pBase);
  printf("Direccion del puntero pBase es: %p \n", &pBase);
  return 0;
}
```

```
Valor de base es: 5
Direccion de base es: 0000
Valor del puntero pBase es: 0000
Valor que apunta el puntero pBase es: 5
Direccion del puntero pBase es: 1000
```

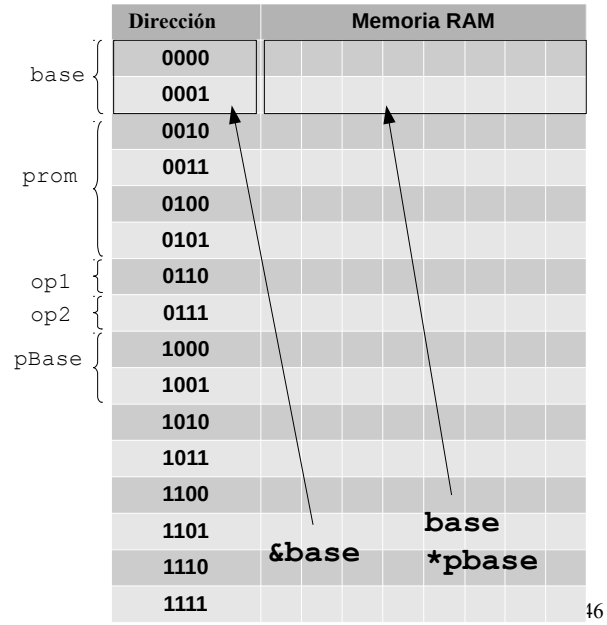
0x1001bf018

0x1001bf020

Direcciones reales de memoria

Tipos de pasaje de parámetros en C
Pasaje por Resultado y Valor/Resultado

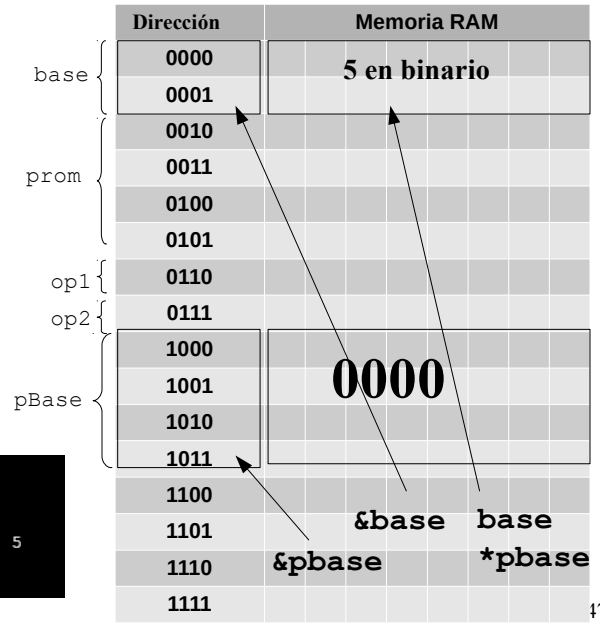
```
#include <stdio.h>
int base; //2 bytes
float prom; //4 bytes
char op1, op2; //1 byte c/u
int *pBase; //4 bytes
int main(){
  base= 5;
  pBase = &base;
  printf("Valor de base es: %d
\n", base);
  printf("Direccion de base
es: %p \n", &base);
  printf("Valor del puntero
pBase es: %p \n", pBase);
  printf("Valor que apunta el
puntero pBase es: %d \n",
*pBase);
  printf("Direccion del
puntero pBase es: %p \n",
&pBase);
  return 0;
}
```



Tipos de pasaje de parámetros en C
Pasaje por Resultado y Valor/Resultado

```
#include <stdio.h>
int base; // 2 bytes
float prom; //4 bytes
char op1, op2; //1 byte c/u
int *pBase; // 4 bytes (en 32bits)
int main(){
  base= 5;
  pBase = &base;
  printf("Valor de base es: %d \
n", base);
  printf("Direccion de base es: %p
\n", &base);
  printf("Valor del puntero pBase
es: %p \n", pBase);
  printf("Valor que apunta el
puntero pBase es: %d \n", *pBase);
  printf("Direccion del puntero
pBase es: %p \n", &pBase);
return 0;
}
```

```
Valor de base es: 5
Direccion de base es: 0000
Valor del puntero pBase es: 0000
Valor que apunta el puntero pBase es: 5
Direccion del puntero pBase 1000
```



Acciones y funciones con parámetros
Ejemplo

```
#include <stdio.h>
int s1, s2, resultado;
int funcionSuma(int a, int b);
void accionSuma(int x, int y, int *res);
int main(){
  printf("Ingrese el primer valor para sumar: ");
  scanf("%i",&s1);
  printf("Ingrese el segundo valor para sumar: ");
  scanf("%i",&s2);
  printf("Llamamos a funcionSuma (pasaje por valor) \n");
  printf("Valor de s1: %i, valor de s2: %i\n",s1,s2);
  resultado = funcionSuma(s1,s2);
  printf("Resultado de la funcionSuma: %i\n",resultado);
  printf("Llamamos a la accionSuma con s1 y s2, ");
  printf("y el resultado lo guardamos usando un puntero a entero llamado res \n");
  printf("Valor de s1: %i, valor de s2: %i\n",s1,s2);
  accionSuma(s1,s2,&resultado);
  printf("Resultado de la accionSuma: %i\n",resultado);
  printf("Valor de la posicion en memoria de la variable resultado: %p\n",&resultado);
  return 0;
  }
int funcionSuma(int a, int b){
  return a+b;
}
void accionSuma(int x, int y, int *res){
  *res = x+y;
}
```

Acciones y funciones con parámetros
Ejemplo

```
#include <stdio.h>
int s1, s2, resultado;
int funcionSuma(int a, int b);
void accionSuma(int x, int y, int *res);
int main(){
  printf("Ingrese el primer valor para sumar: ");
  scanf("%i",&s1);
  printf("Ingrese el segundo valor para sumar: ");
  scanf("%i",&s2);
  printf("Llamamos a funcionSuma (pasaje por valor) \n");
  printf("Valor de s1: %i, valor de s2: %i\n",s1,s2);
  resultado = funcionSuma(s1,s2);
  printf("Resultado de la funcionSuma: %i\n",resultado);
  printf("Llamamos a la accionSuma con s1 y s2, ");
  printf("y el resultado lo guardamos usando un puntero a entero llamado res \n");
  printf("Valor de s1: %i, valor de s2: %i\n",s1,s2);
  accionSuma(s1,s2,&resultado);
  printf("Resultado de la accionSuma: %i\n",resultado);
  printf("Valor de la posicion en memoria de la variable resultado: %p\n",&resultado);
  return 0;
}
int funcionSuma(int a, int b){
  return a+b;
}
void accionSuma(int x, int y, int *res){
  *res = x+y;
}
```

```
Ingrese el primer valor para sumar: 2
Ingrese el segundo valor para sumar: 7
Llamamos a funcionSuma (pasaje por valor)
Valor de s1: 2, valor de s2: 7
Resultado de la funcionSuma: 9
Llamamos a la accionSuma con s1 y s2, y el resultado lo guardamos usando un puntero a entero llamado res
Valor de s1: 2, valor de s2: 7
Resultado de la accionSuma: 9
Valor de la posicion en memoria de la variable resultado: 0x601050
```


Parámetros
Tipos de pasaje de parámetros

Notación Algorítmica	C
dato	
dato-resultado	* /&
resultado	* /&

C no los tiene, es nada más que una forma de simularlo.

Acciones con parámetros
Ejemplo

Notación Algorítmica

```
Acción CalcularAreaTriángulo(dato baseTri ∈ R, alturaTri ∈ R,
                             resultado areaTri ∈ R)
Inicio
  areaTri ← (baseTri*alturaTri)/2
Facción
```

C

```
void CalcularAreaTriangulo(float baseTri, float alturaTri, float *areaTri){
    *areaTri = (baseTri*alturaTri)/2;
}

Invocación: CalcularAreaTriangulo(base, altura, &area);
```


Acciones con parámetros
Ejemplo

Notación Algorítmica

```
Acción InformarResultados(dato a ∈ R)
Inicio
    Salida:a
Facción
```

C

```
void InformarResultados(float a){
    printf("El area del triangulo es: %f \n",a);
}

Invocación: InformarResultados(area);
```

Acciones con parámetros
Ejemplo
Notación Algorítmica

```
Acción ObtenerDatos(resultado x ∈ R, y ∈ R)
Inicio
    Entrada:x, y
Facción
```

C

```
void ObtenerDatos(float *x, float *y){
    printf("Ingrese la base: \n ");
    scanf("%f",&(*x));
    printf("Ingrese el altura: \n");
    scanf("%f",&(*y));
}

Invocación: ObtenerDatos(&base, &altura);
```


Acciones con parámetros
Ejemplo

```
#include <stdio.h>
float base, altura; //base y altura del triangulo
float area;         //area del triangulo
void ObtenerDatos(float *x, float *y){
    printf("Ingrese la base: \n ");
    scanf("%f",&(*x));
    printf("Ingrese el altura: \n");
    scanf("%f",&(*y));
}
void CalcularAreaTriangulo(float baseTri, float alturaTri, float
*areaTri){
    *areaTri = (baseTri*alturaTri)/2;
}
void InformarResultados(float a){
    printf("El area del triangulo es: %f \n",a);
}
int main(){
    ObtenerDatos(&base, &altura);
    CalcularAreaTriangulo(base, altura, &area);
    InformarResultados(area);
    return 0;
}
```


Acciones con parámetros
Ejemplo (perfiles)

```
#include <stdio.h>
float base, altura;     //base y altura del triangulo
float area;    //area del triangulo
void ObtenerDatos(float *x, float *y);
void CalcularAreaTriangulo(float baseTri, float alturaTri, float *areaTri);
void InformarResultados(float a);
int main(){
    ObtenerDatos(&base, &altura);
    CalcularAreaTriangulo(base, altura, &area);
    InformarResultados(area);
    return 0;
}
void ObtenerDatos(float *x, float *y){
    printf("Ingrese la base: \n ");
    scanf("%f",&(*x));
    printf("Ingrese el altura: \n");
    scanf("%f",&(*y));
}
void CalcularAreaTriangulo(float baseTri, float alturaTri, float *areaTri){
    *areaTri = (baseTri*alturaTri)/2;
}
void InformarResultados(float a){
    printf("El area del triangulo es: %f \n",a);
}
```


Acciones con parámetros
Ejemplo

Notación Algorítmica

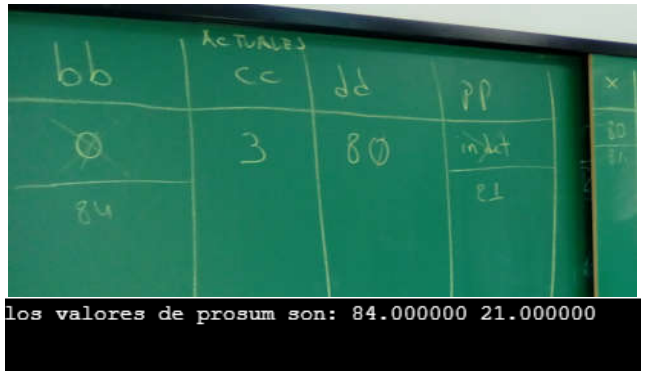
```
Acción ProSum(dato x,z ∈ R, dato-resultado w ∈ R, resultado prom ∈ R)
Léxico local
  a ∈ R
Inicio
  x ← x+1
  a ← w*2
  w ← x+z+w+a
  z ← z / 2
  prom ← w / 4
Facción
```

C

```
void ProSum(float x, float z, float *w, float *prom){
  float a;

  x = x+1;
  a = (*w)*2;
  *w = x+z+(*w)+a;
  z = z/2;
  *prom = (*w)/4;
}
```

Acciones con parámetros
Ejemplo

```
#include <stdio.h> //ProSumEjemplo1
float bb, cc, dd, pp;
void ProSum(float x, float z, float *w, float *prom){
  float a;

  x = x+1;
  a = (*w)*2;
  *w = x+z+(*w)+a;
  z = z/2;
  *prom = (*w)/4;
}

int main()
{
    dd=80;
    cc=3;
    bb=0;
    ProSum(dd,cc,&bb,&pp);
    printf("los valores de prosum son: %f %f \n",bb,pp);
    return 0;
}
```



Bibliografía

– Watt, David: Programming Language Concepts and Paradigms, Prentice-Hall International Series in Computer Science (1990). Cap. 5 (pags. 116-129)

– Biondi, J. y Clavel, G. “Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes”: (pags. 181 - 190)

– Scholl, P. y Peyrin, J.-P. “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”. (pags. 71 - 87)

– Quetglás, Toledo, Cerverón. "Fundamentos de Informática y Programación“. Capítulo 3. http://robotica.uv.es/Libro/Indice.html

- Programación Modular (pags 110 - 111)

– Kernighan, B; Ritchie, D. El lenguaje de programación C. Pearson Educación, 1991. Capp. 1 (pags. 26-29), Cap. 5 (pags. 103-108)

– Rancel, M. “Curso básico de programación lenguaje C desde cero”. Capítulo 15. PROGRAMACIÓN POR MÓDULOS CON C. https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=947:funciones-en-c-ique-significa-void-ique-es-el-tipo-de-retorno-ipara-que-sirve-return-modulos-cu00547f&catid=82&Itemid=210