

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 4

Modularización, Abstracción Funciones



2018 Lic. Ariel Ferreira Szpiniak

Noticias

24 de marzo de 1976 – 24 de marzo de 2018
Hacer memoria es también hacer justicia

Berta Perassi (estudiante), Ernesto Silber (docente), José Peco Duarte (No docente), Jorge R. Harriague (estudiante y miembro de la Comisión fundadora de la UNRC), Federico Harriague (estudiante), José Pocho Amato (estudiante) y Alberto Pinto (docente) fueron miembros de nuestra comunidad universitaria, víctimas de la dictadura militar.

En la ciudad de Río Cuarto y zona se registran 172 de detenidos por razones políticas y 40 desaparecidos.



2018 Lic. Ariel Ferreira Szpiniak

2

Noticias

24 de marzo de 1976 – 24 de marzo de 2018
Hacer memoria es también hacer justicia

Frente al Comedor universitario hay una baldosa en memoria del profesor **Ernesto Silber** secuestrado dentro de la UNRC el 9 de agosto de 1976. Otra baldosa se colocó en la vereda de la **Unidad Regional 9** (Policía de la Provincia de Córdoba), Belgrano 58, donde **Ernesto Silber** fue **torturado** y apareció **muerto** dentro de una celda 3 días después.

En reconocimiento a su entrega y por decisión del Concejo Deliberante, en 2007, en el Barrio Las Delicias de la ciudad de Río Cuarto tuvo lugar la inauguración del Pasaje Público Alfabetizadora **Berta Perassi**.

En Villa Mercedes, en 2016, una Plaza fue bautizada con el nombre de los hermanos **Helena María y Federico Harriague**, desaparecidos en la última dictadura.

En la Escuela Escuela Bartolomé Mitre, donde fue alumno **Peco Duarte**, el jueves pasado se descubrió una placa conmemorativa en su nombre.



2018 Lic. Ariel Ferreira Szpiniak

3

Noticias

Dictadura cívico-militar 1976-1983:
FUE TERRORISMO DE ESTADO

- **610** Centros Clandestinos de Detención, Tortura y Exterminio (en Córdoba 59). 610 en 1976 (muchos temporales y circunstanciales, 364 estables).
- **30.000** desaparecidos
- **500** bebés robados (**127** recuperados).
- Miles de **exiliados** (artistas, músicos, cantores, científicos, militantes políticos, gremialistas, docentes, trabajadores en general, estudiantes)
- Listas **negras** (Norma Aleandro, Héctor Alterio, Luis Brandoni, Norman Briski, Nacha Guevara, Víctor Laplace y Federico Luppi, Eduardo Galeano, Julio Cortázar, David Viñas, María Elena Walsh, Osvaldo Bayer, Rogelio García Lupo, Tomas Eloy Martínez, Francisco "Paco" Urondo, Jacobo Timerman, Leonardo Favio, Miguel Ángel Estrella, Jaime Dávalos, Los Trovadores, Horacio Guarany, Víctor Heredia, Armando Tejada Gómez, Osvaldo Pugliese, Mercedes Sosa, Antonio Berni, Leonardo Favio, Octavio Getino, Fernando "Pino" Solanas.



2018 Lic. Ariel Ferreira Szpiniak

4

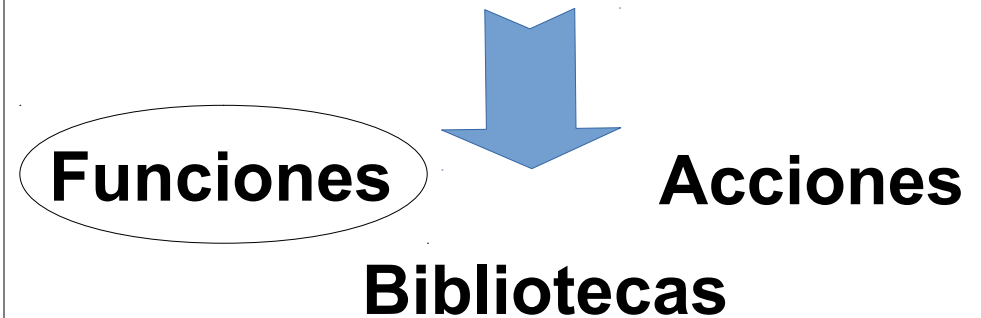
Noticias

Diactadura cívico-militar 1976-1983: FUE TERRORISMO DE ESTADO

- **Canciones** (más de 150), **artistas** (León Gieco, Charly García, Luis Alberto Spinetta, Cacho Castaña, Horacio Guarany, Ramón "Palito" Ortega, Eric Clapton, Pink Floyd, Queen), y **libros prohibidos**, ("Un elefante ocupa mucho espacio", "Mascaró, el cazador americana" (Haroldo Conti), "El Principito", "1001 sueños eróticos", "Cuarteles de invierno" y "No habrá más penas ni olvido", (Osvaldo Soriano), "Variaciones en rojo" y "Operación Masacre" (Rodolfo Walsh), **casas** y **empresas saqueadas**.
- **Congreso** de la Nación, **Legislaturas** Provinciales, y **Consejos Deliberantes disueltos**.
- **Remoción** de jueces de la **Corte Suprema**, los los **Tribunales Superiores Provinciales**, y **Procurador General**.
- **Sindicatos intervenidos**
- **Supresión** del **derecho a huelga**



Modularización Abstracción



Modularización

El arte de la programación es el arte de organizar la complejidad. Dijkstra - 1972

Los problemas del mundo real se caracterizan por su:

- Complejidad
- Extensión
- Variación en el tiempo (modificaciones)

Los tratamos de resolver empleando:

- Abstracción.
- Descomposición.
- Independencia Funcional.

Modularizar significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos.

No se trata simplemente de subdividir el código de un sistema de software en bloques con un número de instrucciones dado, sino de separar funciones lógicas con datos propios y datos de comunicación perfectamente especificados.



Modularización Ventajas

▪ Productividad

Al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad.

▪ Reusabilidad

Una cuestión central en el desarrollo de software es la reusabilidad, es decir la posibilidad de utilizar repetidamente un programa o parte del mismo. La descomposición que ofrece la modularización favorece el reuso.

▪ Mantenimiento correctivo

La división lógica de un programa en módulos permite encontrar los errores que se producen con mayor facilidad. Esto significa poder corregir los errores en menor tiempo y disminuir los costos de mantenimiento de los programas.



Modularización

Ventajas

▪ Facilidades de crecimiento

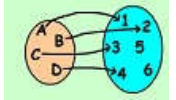
Los problemas reales crecen, es decir, aparecen con el tiempo nuevos requerimientos del usuario. La modularización permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un programa en funcionamiento.

▪ Mejor legibilidad

Un efecto de la modularización es una mayor claridad para leer y comprender el código del programa.



Concepto de función



- **Idea intuitiva:** Una función establece una asociación entre valores de entrada y valores de salida.
- **Definición:** Una función es una correspondencia elementos de dos conjuntos, en la que un elemento del conjunto origen (dominio), le corresponde un único elemento del conjunto imagen (rango).

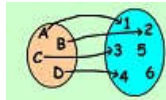
$$\forall x_1, x_2 \in D: x_1 = x_2 \rightarrow f(x_1) = f(x_2)$$

Ejemplo: Función doble

- dominio: números enteros
- rango : números enteros



Concepto de función



- **Función parcial:** Son aquellas en las que la imagen de, al menos, un elemento no está definida.

$$\exists x \in D: \exists y \notin I: f(x) = y$$

- Ejemplo: división entera

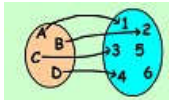
- **Función total:** Son aquellas en las que todos los elementos del dominio tienen definida una imagen.

$$\forall x \in D: \exists y: y \in I: f(x) = y$$

- Ejemplo: valor absoluto



Concepto de función



- **Definición de funciones:**

- Intensión
- Extensión

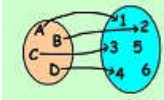
- **Ejemplo:** función doble

- Definición por extensión: $\{ \dots (-2, -4), (-1, -2), (0, 0), (1, 2), (2, 4) \dots \}$
- Definición por intensión: $\text{doble}(x) = x + x$

- **La definición de una función puede combinar ambas formas de definición.**



Concepto de función



- **Aplicación:** particularización de la regla de correspondencia a un valor concreto de dominio que determina un valor concreto de la imagen.

– Ejemplo: $\text{doble}(5) = 10$

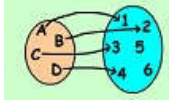
- Al valor del conjunto dominio se lo denomina argumento o entrada de la función.
- El valor del conjunto imagen es el resultado o salida de la función.

doble: nombre de la función
5: argumento o entrada
10: resultado o salida

- La **aplicación** sobre un valor desencadena una secuencia de sustituciones hasta llegar a la expresión más reducida posible que se conoce con el nombre de **expresión canónica o forma normal**.



Concepto de función



- **Composición:** El argumento o entrada de una función puede provenir de la salida de otra aplicación. Posibilita el anidamiento de aplicaciones.



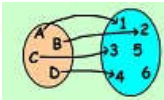
– Ejemplos:

$\text{doble}(\text{doble}(5)) = 20$ \longrightarrow $\boxed{\text{doble}} \longrightarrow \boxed{\text{doble}} \longrightarrow$

$\text{doble}(\text{mayor}(7,4)) = 14$ $\searrow \rightarrow \boxed{\text{mayor}} \longrightarrow \boxed{\text{doble}} \longrightarrow$

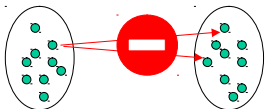


Concepto de función



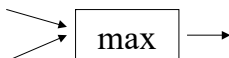
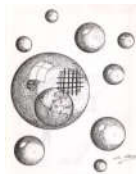
Propiedades de interés computacional

- **Determinismo:** Dado un argumento de entrada, una función siempre devuelve el mismo resultado.



- **Dependencia de los argumentos:**

El resultado devuelto por una función *sólo* depende de sus argumentos de entrada.



Abstracción Funciones

- Una función representa la evaluación de una expresión.
- Cuando una función es evaluada, ésta devuelve como resultado un único valor.
- Una función total asocia cada elemento del rango a un elemento del dominio:

$$f : A \rightarrow B$$

indica que: $f \subseteq A \times B$

$$\forall x_1, x_2 \in D: x_1 = x_2 \rightarrow f(x_1) = f(x_2)$$

$$\forall x \in D: \exists y \in I: f(x) = y$$

- Son similares a las funciones matemáticas.



Abstracción

Funciones

- Poseen las mismas ventajas que las acciones.
- Pueden, y deben, utilizarse en **asignaciones**, **comparaciones** o **expresiones**.
- **No** deben **modificar** el **entorno**.
- Las funciones **devuelven** un **valor** si o si.
- No deben tener **EFFECTOS COLATERALES!!!**
- Los efectos colaterales pueden producir que la función deje de ser **determinística**.
- Una función se puede pensar como un valor que es del tipo de su resultado.



Funciones

Estructura

- Se componen de:
 1. Un encabezamiento: el cual tiene la palabra reservada **Función** seguida de un identificador, los parámetros de entrada (opcional), una \rightarrow y el tipo de dato de lo que devuelve.
 2. Declaraciones locales: esto es un léxico local donde se declaran las variables locales a la función.
 3. Bloque o Cuerpo de acciones ejecutables: encerradas entre **Inicio** y **Ffuncion** se desarrollan el conjunto de acciones o composiciones (secuenciales, condicionales, etc.) que resuelven la especificación de la función. En, al menos, una acción debe devolver un resultado. Para ello se utiliza la \leftarrow .



Funciones

Estructura

1. *Cabecera* **Función** <identificador>(<lista de parámetros>) \rightarrow tipo
2. *Declaraciones* Lexico local (si es necesario)
variables, constantes, etc.
3. *Sentencias ejecutables* Inicio
`<acción más simple>`
`...`
`<acción más simple>`
`{en al menos una acción debe devolver`
`un resultado. Para ello se utiliza`
`la \leftarrow de la asignación (que indica`
`que es lo que devuelve la función)}`
Ffuncion
4. *Ubicación*
Las declaraciones de funciones se hacen en el léxico del algoritmo principal, después de las declaraciones de los identificadores del mismo (variables, etc.).



Funciones

Invocación y ejecución

- Una función se ejecuta indicando su nombre y los parámetros. Esto se conoce como “llamado” o “invocación” de la función. El resultado es lo que devuelve la función.
- Luego de invocar a una función, y recibir el resultado, el algoritmo continúa.



Funciones con parámetros

- Las funciones 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.



Funciones con parámetros

- Si las funciones que dan solución a los subproblemas son módulos **autocontenidos**, uno puede resolver y testear cada función **independientemente** del resto.
- Para que las funciones sean **autocontenidas** **no deben** hacer **referencia** **declaraciones** (variables, constantes, tipos) que estén **fuera de dicha función**, como por ejemplo el léxico del algoritmo principal.



Funciones con parámetros

- Para lograr que una función pueda ser considerada **autocontenida**, la información debe poder ser transferida entre ella y el resto del algoritmo principal a través de lo que llamaremos **parámetros**.
- Los parámetros permiten que una función pueda manipular diferentes valores, y por lo tanto la misma función puede ser usada tantas veces como sea necesario en un mismo algoritmo.



Funciones con parámetros

Motivación

Ejemplo: Mayor de tres números

Supongamos que deseamos resolver el siguiente problema:

Encontrar el menor valor entre tres números enteros distintos entre sí.

Pensemos una solución.....

Un algoritmo que solucione el problema planteado puede ser:



Motivación

~~Función menorDeTresNumeros $\rightarrow \mathbb{Z}$~~
~~Inicio~~
~~según~~
~~(p < s y p < t): \leftarrow p~~
~~(s < p y s < t): \leftarrow s~~
~~(t < p y t < s): \leftarrow t~~
~~fsegún~~
~~Ffuncion~~

¿Cómo independizamos la entrada de datos?

Podemos agregar *parámetros* y darles un nombre y tipo a cada uno:

Función menorDeTresNumeros (i, j, k $\in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio
 según
 (i < j y i < k): \leftarrow i
 (j < i y j < k): \leftarrow j
 (k < i y k < j): \leftarrow k
 fsegún
 Ffuncion



Tipos de parámetros

Parámetros Formales

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

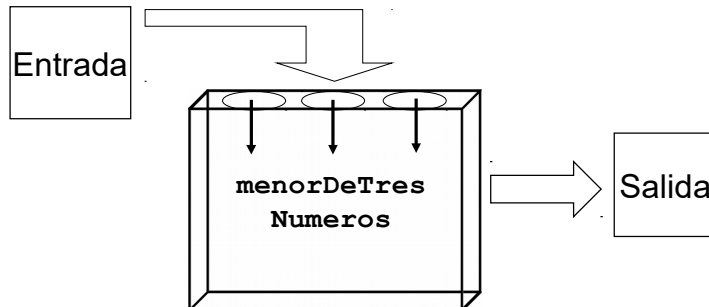
Parámetros Actuales

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



Tipos de parámetros

- Los parámetros listados en la cabecera de una función son llamados parámetros formales. Ellos sirven como “agujeros” a ser llenados por los calores pasados como parámetros actuales (valores reales) cuando la función es invocada.



Acciones con parámetros

Invocación o llamado - Ejemplo

Algoritmo elMenor

Lexico

p, s, t $\in \mathbb{Z}$ //datos de entrada
 aux $\in \mathbb{Z}$ //variable auxiliar

Función menorDeTresNumeros (i, j, k $\in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según

(i < j y i < k): \leftarrow i
 (j < i y j < k): \leftarrow j
 (k < i y k < j): \leftarrow k

fsegún

Ffuncion

Inicio

Entrada: p, s, t
 aux \leftarrow menorDeTresNumeros (p, s, t)
 Salida: aux

Fin

Tipo del resultado devuelto por la función

Parámetros Formales
i, j, k

Parámetros Actuales
p, s, t

¿Cómo hacemos para indicar que i, j, k son parámetros de entrada?



Tipos de pasaje de parámetros

- Existen diversos enfoques y clasificaciones sobre los tipos de pasajes de parámetros.
- Nosotros realizaremos una clasificación lo más general posible, “ideal” desde el punto de vista algorítmico, pero donde posiblemente no encontremos una traducción “directa” en los lenguajes de programación.
- Al igual que el “según”, debemos analizar las características del lenguaje donde voy a implementar mis algoritmos a los efectos de tomar las decisiones que correspondan para poder traducirlo.



Tipos de pasaje de parámetros

Por Copia

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

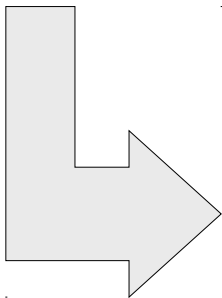
Por Referencia

vincula el parámetro formal directamente al argumento en sí.



Tipos de pasaje de parámetros

Por COPIA

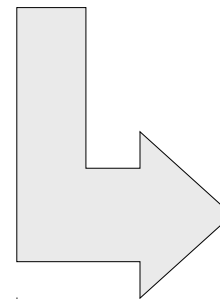


- **Entrada: Por Valor**
- **Salida: Por Resultado**
- **Entrada/Salida: Por Valor/Resultado**



Tipos de pasaje de parámetros

Por Referencia



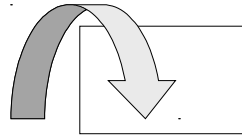
- Es un mecanismo de pasaje de parámetros permite que el parámetro formal sea ligado directamente al argumento mismo (parámetro actual).
Aparecen bajo varios “disfraces”, o simulados, en algunos lenguajes de programación.



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.

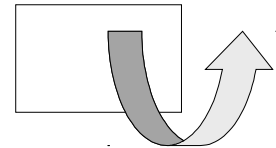
Al asociarse el parámetro formal (puede ser una variable o una constante) sólo al valor inicial del parámetro actual, las modificaciones en el parámetro formal no afectan al parámetro actual.



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*.

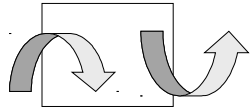
Cuando el módulo finaliza su ejecución, el valor final del parámetro formal se asocia al parámetro actual, es decir se le asigna un resultado a la variable utilizada durante la invocación.



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.



Tipos de pasaje de parámetros

- Los parámetros por valor (cláusula **dato**) pueden ser valores concretos (4, 8, True, 's', 'w', etc), constantes, variables, expresiones o invocaciones a funciones .
- Los parámetros por resultado o valor/resultado (cláusula **resultado** y **dato-resultado**) solo pueden ser variables pues en ellos se debe alojar un valor si o si.



Funciones con parámetros

Invocación o llamado - Ejemplo

Algoritmo elMenor

Lexico

$p, s, t \in \mathbb{Z}$ //datos de entrada
 $aux \in \mathbb{Z}$ //variable auxiliar

Función menorDeTresNumeros (dato $i, j, k \in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según

$(i < j \text{ y } i < k): \leftarrow i$
 $(j < i \text{ y } j < k): \leftarrow j$
 $(k < i \text{ y } k < j): \leftarrow k$

fsegún

Ffuncion

Inicio

Entrada: p, s, t
 $aux \leftarrow \text{menorDeTresNumeros}(p, s, t)$
 Salida: aux

Fin

Las funciones deben usar pasaje por valor (cáusula dato), para garantizar que no se modifique el entorno al ser invocadas.

Tipo del resultado devuelto por la función

Parámetros Formales
 i, j, k

Parámetros Actuales
 p, s, t



Funciones con parámetros

Invocación o llamado - Ejemplo

Algoritmo elMenor

Lexico

$p, s, t \in \mathbb{Z}$ //datos de entrada
 $aux \in \mathbb{Z}$ //variable auxiliar

Función menorDeTresNumeros (dato $i, j, k \in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según

$(i < j \text{ y } i < k): \leftarrow i$
 $(j < i \text{ y } j < k): \leftarrow j$
 $(k < i \text{ y } k < j): \leftarrow k$

fsegún

Ffuncion

Inicio

Entrada: p, s, t
 Salida: $\text{menorDeTresNumeros}(p, s, t)$

Fin

Las funciones deben usar pasaje por valor (cáusula dato), para garantizar que no se modifique el entorno al ser invocadas.

Tipo del resultado devuelto por la función

Parámetros Formales
 i, j, k

Parámetros Actuales
 p, s, t



Abstracción

Funciones

¿Dónde las usamos?

• En asignaciones:

...
 $\text{resultado1} \leftarrow \text{max2}(\text{var1}, \text{var2})$
 ...
 $\text{resultado2} \leftarrow \text{max2}(\text{var3}, 45)$
 ...

• En comparaciones:

...
si $(\text{max2}(\text{nota1}, \text{nota2}) < 6)$
entonces
fsi ...

• En expresiones:

...
 $\text{duploMayor} \leftarrow \text{max2}(z1, z2) * 2$
 ...
si $((\text{max2}(z3, z4) + 1) > 8)$
entonces
fsi ...



Funciones con parámetros

Invocación en asignación

Algoritmo elMenor

Lexico

$p, s, t \in \mathbb{Z}$ //datos de entrada
 $aux \in \mathbb{Z}$ //variable auxiliar

Función menorDeTresNumeros (dato $i, j, k \in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según

$(i < j \text{ y } i < k): \leftarrow i$
 $(j < i \text{ y } j < k): \leftarrow j$
 $(k < i \text{ y } k < j): \leftarrow k$

fsegún

Ffuncion

Inicio

Entrada: p, s, t
 $aux \leftarrow \text{menorDeTresNumeros}(p, s, t)$
 Salida: aux

Fin

Las funciones deben usar pasaje por valor (cáusula dato), para garantizar que no se modifique el entorno al ser invocadas.

Tipo del resultado devuelto por la función

Parámetros Formales
 i, j, k

Parámetros Actuales
 p, s, t



Funciones con parámetros

Invocación en una expresión

Algoritmo elMenor

Lexico

$p, s, t \in \mathbb{Z}$ //datos de entrada
 $aux \in \mathbb{Z}$ //variable auxiliar
 $auxLogica \in \text{Logico}$ //variable auxiliar

Función menorDeTresNumeros (dato $i, j, k \in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según
 $(i < j \text{ y } i < k) : \leftarrow i$
 $(j < i \text{ y } j < k) : \leftarrow j$
 $(k < i \text{ y } k < j) : \leftarrow k$

fsegún

Ffuncion

Inicio

Entrada: p, s, t
 $aux \leftarrow \text{menorDeTresNumeros}(p, s, t)$
 Salida: aux
 $aux \leftarrow (aux + \text{menorDeTresNumeros}(p, s, t))$
 Salida: aux
 $aux \leftarrow (\text{menorDeTresNumeros}(p, s, t) * 2)$
 Salida: aux
 $auxLogica \leftarrow (\text{menorDeTresNumeros}(p, s, t) \geq 0)$
 Salida: $auxLogica$

Fin

Tipo del resultado
devuelto por la
función

Parámetros
Formales
 i, j, k

Parámetros
Actuales
 p, s, t



2018 Lic. Ariel Ferreira Szpiniak

41

Funciones con parámetros

Invocación en una expresión

Algoritmo elMenor

Lexico

$p, s, t \in \mathbb{Z}$ //datos de entrada
 $aux \in \mathbb{Z}$ //variable auxiliar

Función menorDeTresNumeros (dato $i, j, k \in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según
 $(i < j \text{ y } i < k) : \leftarrow i$
 $(j < i \text{ y } j < k) : \leftarrow j$
 $(k < i \text{ y } k < j) : \leftarrow k$

fsegún

Ffuncion

Inicio

Entrada: p, s, t
 $aux \leftarrow \text{menorDeTresNumeros}(p, s, t)$
 Salida: aux
 Salida: $(aux + \text{menorDeTresNumeros}(p, s, t))$

Fin

Tipo del resultado
devuelto por la
función

Parámetros
Formales
 i, j, k

Parámetros
Actuales
 p, s, t



2018 Lic. Ariel Ferreira Szpiniak

42

Funciones con parámetros

Invocación en una comparación

Algoritmo elMenor

Lexico

$p, s, t \in \mathbb{Z}$ //datos de entrada
 $\text{resultado} \in \text{Cadena}$

Función menorDeTresNumeros (dato $i, j, k \in \mathbb{Z}$) $\rightarrow \mathbb{Z}$

Inicio

según
 $(i < j \text{ y } i < k) : \leftarrow i$
 $(j < i \text{ y } j < k) : \leftarrow j$
 $(k < i \text{ y } k < j) : \leftarrow k$

fsegún

Ffuncion

Inicio

Entrada: p, s, t
según
 $(\text{menorDeTresNumeros}(p, s, t) \geq 0) : \text{resultado} \leftarrow \text{"El mayor es positivo"}$
 otros: $\text{resultado} \leftarrow \text{"El mayor es negativo"}$
fsegún
 Salida: resultado

Fin

Tipo del resultado
devuelto por la
función

Parámetros
Formales
 i, j, k

Parámetros
Actuales
 p, s, t



2018 Lic. Ariel Ferreira Szpiniak

43

Abstracción

Funciones

Ejemplo:

$$\text{abs}: \mathbb{R} \rightarrow \mathbb{R}$$

$$\text{abs}(x) = \begin{cases} x, & \text{si } x \geq 0 \\ -x, & \text{sino} \end{cases}$$

Notación algorítmica:

Función abs (dato $x \in \mathbb{R}$) $\rightarrow \mathbb{R}$

Inicio

según

$(x \geq 0) : \leftarrow x$
 $(x < 0) : \leftarrow -x$

fsegún

Ffuncion



2018 Lic. Ariel Ferreira Szpiniak

44

Abstracción

Funciones

Ejemplo

Desarrollar una función que dados dos números reales positivos, calcule y devuelva cual es el mayor de los dos.

Función max2(dato $x, y \in \mathbb{R}$) $\rightarrow \mathbb{R}$

Inicio

según

$(x \geq y) : \leftarrow x$

$(x < y) : \leftarrow y$

fsegún

Ffuncion



Abstracción

Funciones

Ejemplo II

Desarrollar una función que dados tres números reales positivos, calcule y devuelva cual es el mayor de los tres.

Función max3(dato $x, y, z \in \mathbb{R}$) $\rightarrow \mathbb{R}$

Inicio

según

$(x \geq y \text{ y } x \geq z) : \leftarrow x$

$(y > x \text{ y } y > z) : \leftarrow y$

$(z > x \text{ y } z > y) : \leftarrow z$

fsegún

Ffuncion



Abstracción

Funciones

Ejemplo III

Desarrollar una función que dados dos números reales positivos, calcule y devuelva el promedio de los dos.

Función promedio(dato $w, y \in \mathbb{R}$) $\rightarrow \mathbb{R}$

Inicio

$\leftarrow (w+y)/2$

Ffuncion



Abstracción

Funciones

Ejemplo IV

¿Qué hace la siguiente función?

Función EsMayuscula(dato $c \in \text{Caracter}$) $\rightarrow \text{Lógico}$

Inicio

$\leftarrow (c \geq 'A') \text{ y } (c \leq 'Z')$

Ffuncion



Abstracción

Funciones

Ejemplo V

¿Qué hace la siguiente función?

Función `EsVocal(dato q ∈ Caracter) → Lógico`

Léxico local

`minus ∈ Caracter`

Inicio

```
minus ← AMinuscula(q)
← (minus = 'a') o (minus = 'e') o
   (minus = 'i') o (minus = 'o') o (minus = 'u')
```

Ffuncion

Función `AMinuscula`

Es otra función.
Hay que definirla



Funciones en C

Estructura general

Se pueden traducir fácilmente a C

```
tipoQueDevuelve <nombre>(listaDeParámetros) {
    //Léxico propio de la función (opcional)
    <sentencias>;
    return (valorQueDevuelve);
    /*Para indicar lo que devuelve la
    función de colocarse el return.
    Puede haber más de un return (ojo).
    El tipo del valorQueDevuelve debe
    coincidir con el tipoQueDevuelve */
}
```



Funciones en C

Estructura general. Ejemplos

```
int EsMayuscula(char c){
    return ((c >= 'A') && (c <= 'Z'));
}
```

```
char AMinuscula(char c){
    /* si un caracter esta comprendido
    entre A y Z, se le suma la
    diferencia entre los ASCII de las
    minúsculas y las mayúsculas ( 97 -
    65 = 32 ) para a minuscula */
    return (c + ('a'-'A'));
}
```



Funciones en C

Estructura general. Ejemplos

```
int EsVocal(char c){
    char minus; //Léxico local
    if (EsMayuscula(c)){
        minus = AMinuscula(c);
    }
    else{
        minus = c;
    }
    return ((minus == 'a') || minus == 'e')
        || (minus == 'i') || (minus == 'o') ||
        (minus == 'u'));
}
```



Abstracción

Funciones

Composición de Funciones

Es similar a la composición de funciones matemáticas:

La composición matemática $f \circ g(x)$ sería en notación algorítmica $f(g(x))$, es decir, similar a la definición de composición matemática.

```
maximoDeTres    ← max2 (x, max2 (y, z))  
  
maximoParImpar  ← mod (max2 (z1, z2), 2)  
  
promMaximos     ← promedio (max2 (n1, n2), max2 (n3, n4))
```



Abstracción

Funciones

Composición de Funciones

A las composiciones podemos darle nombres. En tal caso serán funciones más complejas.

Función `max3 (dato x, y, z ∈ R) → R`

Inicio

```
← max2 (x, max2 (y, z))
```

Ffuncion



Abstracción

Funciones

Composición de Funciones

Ejemplo

Usando composición de funciones, realizar el promedio entre el número más grande del par (x,y) y del par (z,w):

Función `promEntreMaximos (dato x, y, z, w ∈ R) → R`

Inicio

```
← promedio (max2 (x, y), max2 (z, w))
```

Ffuncion



Abstracción

Funciones

Por extensión

- Las funciones vistas se denominan por intensión, en contraposición con las que veremos ahora en donde determinamos caso por caso que valor del rango le corresponde a cada elemento del dominio.
- Las funciones por extensión no pueden utilizarse para cualquier contexto ya que poseen una serie de restricciones:
 - uno o dos parámetros,
 - tipo de entrada *discreto* (no *continuo*),
 - tipo de entrada razonablemente pequeño.



Abstracción

Funciones - Por extensión

Función díasDeCadaMes(dato m \in [1..12]) \rightarrow [28..31]

Inicio

según

m=1 \circ m=3 \circ m=5 \circ m=7 \circ m=8 \circ m=10 \circ m=12: \leftarrow 31

m=4 \circ m=6 \circ m=9 \circ m=11: \leftarrow 30

m=2: \leftarrow 28

fsegún

Ffuncion



Abstracción

Funciones - Por extensión

Función díasDeLaSemana(dato d \in [1..7]) \rightarrow Dia
Inicio

según

d=1 : \leftarrow Lunes

d=2 : \leftarrow Martes

d=3 : \leftarrow Miercoles

d=4 : \leftarrow Jueves

d=5 : \leftarrow Viernes

d=6 : \leftarrow Sabado

d=7 : \leftarrow Domingo

fsegún

Ffuncion

Donde **Dia** es un tipo Numerado

Dia = (Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo)



Ejemplo Completo de programa en C con funciones

```
#include <stdio.h> // IdentificarCaracter
char letra;
int EsMayuscula(char c) // primera versión usando variable local
int EsMayuscula2(char c) // segunda versión sin variable local
int EsMayuscula3(char c) // tercera versión sin variable local y sin if
char AMinuscula(char c)
void main()
{
    printf("Ingresar caracter: ");
    scanf("%c", &letra);
    if (EsMayuscula(letra))
    {
        printf("El caracter ingresado (%c) es una mayuscula", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una mayuscula", letra);
    }
    if (EsVocal(letra))
    {
        printf("El caracter ingresado (%c) es una vocal", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una vocal", letra);
    }
    if (EsMayuscula2(letra))
    {
        printf("El caracter ingresado (%c) es una mayuscula", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una mayuscula", letra);
    }
    if (EsMayuscula3(letra))
    {
        printf("El caracter ingresado (%c) es una mayuscula", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una mayuscula", letra);
    }
    if (EsVocal(letra))
    {
        printf("El caracter ingresado (%c) es una vocal", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una vocal", letra);
    }
}
```

```
int EsMayuscula(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
int EsMayuscula2(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
int EsMayuscula3(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
char AMinuscula(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return c - 'A' + 'a';
    }
    return c;
}
```

```
void main()
{
    printf("Ingresar caracter: ");
    scanf("%c", &letra);
    if (EsMayuscula(letra))
    {
        printf("El caracter ingresado (%c) es una mayuscula", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una mayuscula", letra);
    }
    if (EsVocal(letra))
    {
        printf("El caracter ingresado (%c) es una vocal", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una vocal", letra);
    }
}
```

```
int EsMayuscula(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
int EsMayuscula2(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
int EsMayuscula3(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
char AMinuscula(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return c - 'A' + 'a';
    }
    return c;
}
```

```
void main()
{
    printf("Ingresar caracter: ");
    scanf("%c", &letra);
    if (EsMayuscula(letra))
    {
        printf("El caracter ingresado (%c) es una mayuscula", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una mayuscula", letra);
    }
    if (EsVocal(letra))
    {
        printf("El caracter ingresado (%c) es una vocal", letra);
    }
    else
    {
        printf("El caracter ingresado (%c) no es una vocal", letra);
    }
}
```

```
int EsMayuscula(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
int EsMayuscula2(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
int EsMayuscula3(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return 1;
    }
    return 0;
}
```

```
char AMinuscula(char c)
{
    if (c >= 'A' && c <= 'Z')
    {
        return c - 'A' + 'a';
    }
    return c;
}
```



Mismo ejemplo en letra más grande

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

```
char letra;
```

```
int EsMayuscula(char c) // primera versión usando variable local
```

```
int EsMayuscula2(char c) // segunda versión sin variable local
```

```
int EsMayuscula3(char c) // tercera versión sin variable local y sin if
```

```
char AMinuscula(char c);
```

```
int EsVocal(char c);
```



Mismo ejemplo en letra más grande

```
void main(){
    printf("\n Ingrese caracter: ");
    scanf("%c",&letra);
    if (EsMayuscula(letra)){
        printf("\n El caracter ingresado (%c) es una
mayuscula", letra);
    }
    else{
        printf("\n El caracter ingresado (%c) es una
minusculta", letra);
    }
    if (EsVocal(letra)){
        printf("\n El caracter ingresado (%c) es una vocal",
letra);
    }
    else{
        printf("\n El caracter ingresado (%c) NO es una vocal",
letra);
    }
}
```



Mismo ejemplo en letra más grande

```
if (EsMayuscula2(letra)){
    printf("\n El caracter ingresado (%c) es una
mayuscula", letra);
}
else{
    printf("\n El caracter ingresado (%c) es una
minusculta", letra);
}
if (EsVocal(letra)){
    printf("\n El caracter ingresado (%c) es una vocal",
letra);
}
else{
    printf("\n El caracter ingresado (%c) NO es una vocal",
letra);
}
}
```



Mismo ejemplo en letra más grande

```
if (EsMayuscula3(letra)){
    printf("\n El caracter ingresado (%c) es una
mayuscula", letra);
}
else{
    printf("\n El caracter ingresado (%c) es una
minusculta", letra);
}
if (EsVocal(letra)){
    printf("\n El caracter ingresado (%c) es una vocal",
letra);
}
else{
    printf("\n El caracter ingresado (%c) NO es una vocal",
letra);
}
}
```



Mismo ejemplo en letra más grande

```
int EsMayuscula(char c){
int aux;
    if ((c >= 'A') && (c <= 'Z')){
        aux=1;
    }
    else{
        aux=0;
    }
    return (aux);
}

int EsMayuscula2(char c){
    if ((c >= 'A') && (c <= 'Z')){
        return(1);
    }
    else{
        return (0);
    }
}
```



Mismo ejemplo en letra más grande

```
int EsMayuscula3(char c){
    return ((c >= 'A') && (c <= 'Z'));
}

char AMinuscula(char c){
    /* si un caracter esta comprendido entre A y Z, se le suma la
    diferencia entre los ASCII de las minúsculas y las
    mayúsculas ( 97 - 65 = 32 ) para a minuscula */
    return (c + ('a'-'A'));
}
```



Mismo ejemplo en letra más grande

```
int EsVocal(char c){
    char minus;
    if (EsMayuscula(c)){
        minus = AMinuscula(c);
    }
    else{
        minus = c;
    }
    return ((minus == 'a') || (minus == 'e') || (minus == 'i')
    || (minus == 'o') || (minus == 'u'));
}
```



Bibliografía

- Watt, David: Programming Language Concepts and Paradigms, Prentice-Hall International Series in Computer Science (1990). Cap. 5
- 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)
- Grogono, P., “Programación en Pascal”, Wilmington, Adisson-Wesley, 1996.
- Wirth, N. and K. Jensen, “Pascal: User Manual and Report”, 4° ed., New York, Springer-Verlag, 1991 (traducción de la primera edición “Pascal: Manual del Usuario e Informe”, Buenos Aires, El Ateneo, 1984).



Citar/Atribuir: Ferreira, Szpiniak, A. (2018). Teoría 4: Modularización, Abstracción. Funciones. Introducción a la Algorítmica y Programación (3300). Departamento de Computación. Facultad de Cs. Exactas, Fco-Qcas y Naturales. Universidad Nacional de Río Cuarto.

Usted es libre para:

Compartir: copiar y redistribuir el material en cualquier medio o formato. Adaptar: remezclar, transformar y crear a partir del material.

El licenciente no puede revocar estas libertades en tanto usted siga los términos de la licencia.

Bajo los siguientes términos:



Atribución: Usted debe darle crédito a esta obra de manera adecuada, proporcionando un enlace a la licencia, e indicando si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo del licenciente.



Compartir Igual: Si usted mezcla, transforma o crea nuevo material a partir de esta obra, usted podrá distribuir su contribución siempre que utilice la misma licencia que la obra original.

<https://creativecommons.org/licenses/by-sa/2.5/ar/>

