UCA
Universidad de Cádiz

Introducción a la Programación
Grado en Ingeniería Informática

Teoría - Curso 2015-2016

Contenido 2 – Problemas, Algoritmos y Programas

Tema 2.- Problemas, Algoritmos y Programas

2.2.1.- CONCEPTO DE ALGORITMO

- Secuencia de pasos que describen el método para resolver un problema
- La ejecución de un algoritmo implica la ejecución de cada uno de sus pasos
- La resolución de un problema exige el diseño del algoritmo adecuado

2.2.1.- CONCEPTO DE ALGORITMO

Pasos para resolver un problema en un ordenador:

1. Análisis del problema
2. Diseño del algoritmo
3. Codificación
4. Traducción, Validación y Ejecución

El ALGORITMO (paso 1) es independiente de:

- El lenguaje de programación (paso 3)
- El ordenador (paso 4)

2.2.1.- CONCEPTO DE ALGORITMO

REQUISITOS de un ALGORITMO:

- Acciones bien definidas
- Secuencia finita de operaciones en un orden determinado
- Debe acabar en un tiempo finito

Un algoritmo es una secuencia finita de operaciones bien definidas que resuelven un problema, cada una de las cuales requieren una cantidad finita de memoria y se realiza en un tiempo finito.

BONDAD de un ALGORITMO:

Factores que la determinan:

- Tiempo
- Recursos de memoria

2.2.1.- Ejemplos de diseño de un algoritmo

Un cliente realiza un pedido a una fábrica. La fábrica examina la ficha del cliente, para comprobar si este es o no solvente antes de enviar el pedido.

1. Inicio.
2. Leer el pedido.
3. Examinar la ficha del cliente.
4. Si el cliente es solvente, aceptar pedido;
5. en caso contrario, rechazar pedido.
6. Fin.

2.2.1.- Ejemplos de diseño de un algoritmo

Suma de los números pares entre 2 y 1000.

2+4+6+8+...+1000. SUMA y NUMERO

1. Inicio.
2. Inicializar SUMA a 0.
3. Inicializar NUMERO a 2.
4. Sumar NUMERO a SUMA. El resultado será el nuevo valor de SUMA.
5. Incrementar NUMERO en 2 unidades.
6. Si NUMERO <= 1000 ir al paso 4; en caso contrario, escribir el valor de SUMA y terminar el proceso.
7. Fin.

2.2.- Proceso de Creación de un Programa

1. Planteamiento y análisis del problema
    - Definición
    - Representación de los datos
        - Entradas
        - Salidas
2. Diseño del Algoritmo
    - Descomposición en subproblemas
    - Diseño descendente
    - Refinamientos sucesivos
    - Optimización y Depuración
3. Implementación
    - Elección del lenguaje
        - Adaptación al problema
        - Conocimientos
        - Disponibilidad de rutinas
        - Herramientas de desarrollo
        - Entorno de la aplicación
    - Codificación (normas de estilo)
4. Ejecución
    - Traducción
    - Prueba y Depuración

2.3.- Datos, Tipos de Datos y Operaciones Primitivas

- **Datos:** Objetos con los cuáles opera un ordenador

  Datos de entrada → Instrucciones → Datos de salida
- El diseño de la **estructura de datos** es tan importante como el diseño del algoritmo
- **Tipos de datos**:
  - Simples
    - numérico
      - entero
      - real
    - lógico
    - carácter
  - Estructurados

- **2.3.2.- Datos numéricos**

- Enteros: subconjunto finito de los enteros
  (Rango : de –32768 a 32768)
- Reales: subconjunto de los números reales
  Notación exponencial o científica:

  $36752010000000000000 \rightarrow 3.675201 \times 10^{20}$

  $.0000000000302579 \rightarrow 3.02579 \times 10^{-11}$

  36.75201×10^{18} 36.75201 *mantisa* 18 *exponente*

2.3.2.- Datos lógicos (booleano)

Posibles valores:

- Verdadero (true)
- Falso (false)

2.3.3.- Datos tipo carácter

- caracteres alfabéticos (A, B, C, ..., Z) (a, b, c, ..., z)
- caracteres numéricos (1, 2, 3, ..., 9, 0)
- caracteres especiales (+, -, *, /, ^, ., ;, <, > $, ...)

En algunos lenguajes existe el tipo cadena como tipo de dato simple Ej/ “Hola”

2.4.1.- Variables y Constantes

Constante: objeto que contiene un valor que no varía durante la ejecución del programa.

Identificador : nombre que identifica a una constante

Tipo : Determinado por el valor que almacena.

enteras, reales, carácter y lógicas Ej/ 20, 1.44, ‘B’, “Pepe”.

Definición:

```
const
   PI =  3.1416
```

2.4.- Variables y Expresiones

2.4.1.- Variables y Constantes

Variable: objeto que contiene un valor que puede variar durante la ejecución del programa

Identificador : nombre que identifica a la variable. Ej /notas

Tipo : No cambia. Determina el tipo del valor que almacenará la variable y la operaciones que se pueden realizar con ella.

enteras, reales, carácter y lógicas

Definición

var

entero: notas

2.4.2.- La operación de asignación

Se utiliza para darle valor a una variable

Se representa por ←

Formato

Identificador_de_variable ← expresión

Notas ← 8

Notas ← 5

El valor que tuviera la variable antes de la operación de asignación desaparece

Evaluación en dos pasos

1.-Se obtiene el valor de la expresión del lado derecho

2.-Se almacena ese valor en la variable cuyo identificador está a la derecha del operador.

2.4.2.- La operación de asignación

Ejemplos

```
var
entero: x, y
    ......
x ← 2
y ← x + 2
```

```
var
entero : n
........
n ← 2
n ← n + 1
```

2.4.2.- La operación de asignación

Ejemplos

Asignación aritmética

```
var
entero : x
.....
x ← 3 + 4 * 8
```

2.4.2.- La operación de asignación

Ejemplos

Asignación lógica

```
var
logico: x
.....
x ← 8< 5
```

Asignación de cadena

```
var
cadena: cad
.....
cad← “Hola que tal”
```

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

Expresión: combinación de variables, constantes, operadores, nombre de funciones y paréntesis, que indican el orden de cálculo.

Ejemplo: $$((cuadrado(8) * p - 4) / 2 - 3 * 5) + d * 3$$

Según el tipo de datos de los elementos de la expresión, ésta puede ser: *aritmética*, cuyo resultado es de tipo numérico; *lógica*, cuyo resultado es de tipo lógico y *carácter*, siendo su resultado de tipo carácter.

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Expresiones Aritméticas**

Operador	*Significado*
+	suma
-	resta
*	multiplicación
/	división entera
div	división real
mod	resto (módulo)

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Reglas de prioridad para las operaciones aritméticas**:

1. Se evalúan primero todas las operaciones que se encuentran entre paréntesis. Si aparecen paréntesis anidados, es decir unos dentro de otros, se evalúan primero las operaciones dentro de los paréntesis más internos.
2. El orden de precedencia para los operadores aritméticos que aparecen en las operaciones será:
    1. operador unario -
    2. operadores *, /, div, mod (todos tienen la misma prioridad)
    3. operadores +, - (ambos presentan la misma prioridad)
    4. Si en una expresión, encerrada o no entre paréntesis, aparece más de un operador con la misma prioridad se evaluará de izquierda a derecha.

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Ejemplos de evaluación de operaciones aritméticas**:

a) 2 * 5 + 4*3
10 + 4*3
10 + 12
22

b) 3 + 2 *6 – 8 / 2
3 + 12 – 8 / 2
3 + 12 – 4
15 – 4
11

c) –4 / 2 * 2 – 3 + 5
–2 * 2 – 3 + 5
–4 – 3 + 5
–7 + 5
–2

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Expresiones lógicas**:

  Se combinan constantes lógicas, variables lógicas y otras expresiones lógicas, utilizando los *operadores lógicos* y los *operadores relacionales.*

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Expresiones lógicas**:

Operadores relacionales

Comparaciones entre valores de tipo numérico, carácter o lógico, y el resultado será verdadero o falso. Con estos operadores se pueden expresar condiciones en los algoritmos, de las cuales dependerá la realización de ciertas tareas.

Formato general:

expresión1 ***operador_de_relación*** *expresión2*

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Expresiones lógicas**:

Operadores relacionales

Operador	Significado
<	menor que
>	mayor que
=	igual que
≤	menor o igual que
≥	mayor o igual que
≠	distinto de

Ejemplos:

4 > 2 verdadero

(3+1) < (8-3) verdadero

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

Prioridad de los Operadores relacionales

- Las comparaciones de datos de tipo numérico, siguen la ordenación habitual que se emplea en matemáticas.
- Cuando se aplican los operadores relacionales a datos de tipo lógico, la constante *falso* es menor que la constante *verdadero*.
- Si en la expresión aparecen caracteres alfabéticos en minúsculas, éstos siguen la ordenación alfabética: 'a' < 'b' < 'c' < … < 'h' < … < 'z'.
- Si en la expresión aparecen caracteres alfabéticos en mayúsculas, éstos siguen la ordenación alfabética: 'A' < 'B' < 'C' < … < 'H' < … < 'Z'.
- Los caracteres que representan a los dígitos decimales guardan su orden. Es decir: '0' < '1' < '2' < … < '8' < '9'.
- Pero si aparecen caracteres especiales ¿, =, *, +,), /, &, %, $, #, \, ..., entonces habría que consultar el código normalizado.

Estas agrupaciones se encuentran en orden decreciente, de esta forma sería:

'*' < '1' < … < '9' < 'A' < … < 'Z' < 'a' < … < 'z'. Sin embargo, sería recomendable consultar el código de caracteres de su ordenador: **ASCII** *(American Standar Code for Information Interchange)* o bien el **EBCDIC** (*Extended Binary-Coded Decimal Interchange Code*).

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Expresiones lógicas**:

Operadores lógicos

Operador lógico	Significado	Expresión lógica
no	negación de p	**no** p
y	conjunción de p y q	p y q
ó	disyunción de p y q	p **ó** q

Tablas de verdad:

no:

p	**no** p
verdadero	falso
falso	verdadero

2.4.3.- Evaluación de Expresiones. Precedencia de Operadores.

- **Expresiones lógicas**: **Operadores lógicos**

Tablas de verdad:

Y:

p	q	p y q
verdadero	verdadero	verdadero
verdadero	falso	falso
falso	verdadero	falso
falso	falso	falso

p y q es verdadero sólo si p y q son verdadero.

Ó:

p	q	p ó q
verdadero	verdadero	verdadero
verdadero	falso	verdadero
falso	verdadero	verdadero
falso	falso	falso

p ó q son verdadero cuando p, q o ambas son verdadero.

2.4.4.- Operaciones de E/S básicas.

- **Lectura**

  Leer (variable_entrada)

  Ejemplo:

  leer(a) **leerá un valor de la entrada y lo almacenará en la variable a.**

- **Escritura**

  Escribir(cadena)

  Ejemplos:

  escribir("Hola, que tal") **escribirá la cadena: Hola, que tal**

  escribir("El valor de la variable a es: ", a)**, suponiendo que el valor almacenado en la variable *a* sea igual a 3, visualizará: El valor de la variable a es 3.**

Métodos de descripción de algoritmos

- Lenguaje Natural.
- Pseudocódigo.
- Diagrama de Flujo.
- Diagrama de Nassi-Schneiderman.

2.5.1.- Lenguaje Natural

- Similar al lenguaje hablado.
- No sigue ninguna norma ni estructura en su representación.
- No es adecuado por su ambigüedad y falta de precisión.

2.5.2.- Pseudocódigo

- Parecido al lenguaje natural, pero está sujeto a determinadas reglas
- Estructura es similar a la de un programa, lo que facilita su traducción a lenguaje de alto nivel.
- Es un punto intermedio entre el lenguaje natural y el lenguaje de alto nivel.
- Es bastante apropiado si se utiliza un lenguaje de programación estructurado por la similitud que existe entre ellos.
- Esta herramienta de diseño de algoritmos permite la posibilidad de describir tipos de datos, constantes, variables y cualquier tipo de expresión, además de instrucciones de entrada y salida e instrucciones de control.

2.5.- Descripción de algoritmos

2.5.2.- Pseudocódigo. Estructura

```
// Comentarios que faciliten la comprensión del algoritmo, por ejemplo, objetivos
// del algoritmo, datos del programador, datos de entrada y datos de salida.

Algoritmo Nombre_algoritmo
const
    //Sección de definición de constantes
tipos
    //Sección de definición de tipos creados por el programador
var
    //Sección de declaración de variables
 inicio
       //Inicialización de variables
       Acción 1
       Acción 2
            .
            .                    //Cuerpo del algoritmo
       Acción n
 fin_algoritmo
```

2.5.3.- Diagrama de Flujo

- **Claridad en la comprensión**
- **Dificultad en la actualización**
- **Se usan unos símbolos que contienen las instrucciones del algoritmo, estos símbolos se enlazan mediante flechas que indican el flujo, el orden o secuencia en el que deben ir realizándose las operaciones.**
- **Diagrama del Sistema u Organigrama: muestra un esquema de la aplicación según los datos de E/S y los soportes donde se encuentran**
- **Diagrama de Detalles u Ordinograma: Se describe toda la secuencia de operaciones que resuelven el problema. Se debe expresar el Comienzo, el fin, las operaciones y la secuencia.**

2.5.3.- Diagrama de Flujo

- **REGLAS DE CONSTRUCCIÓN DE ORDINOGRAMAS:**
  - **El indicador de comienzo y fin del algoritmo sólo puede aparecer una vez.**
  - **El comienzo del algoritmo debe estar en la parte superior.**
  - **El flujo de información debe aparecer de arriba hacia abajo y de izquierda a derecha.**
  - **Las líneas de flujo no deben cruzarse, para ello se utilizan los conectores o bien para la misma página o bien para páginas distintas.**
  - **No pueden existir cruces de líneas de flujo entre operaciones unidas por conectores, éstas también se unen con conectores.**
  - **No debe hacerse mucho uso de comentarios.**

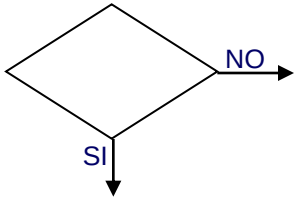
2.5.- Descripción de algoritmos

2.5.3.- Diagrama de Flujo

Terminal. Indica el comienzo (“inicio”) y el final (“fin”) de un algoritmo.

Entrada/Salida. Representa la introducción de datos o bien devolución de resultados.

Proceso. Cualquier tipo de operación entre los datos.



Decisión. Indica operaciones lógicas o relacionales entre los datos

2.5.3.- Diagrama de Flujo

Conector. Se usa para conectar dos puntos cualesquiera de un ordinograma en la misma página. Hay que incluir un conector en la entrada y otro en la salida.

Conector. Se usa para la conexión entre dos puntos cualesquiera de un ordinograma en diferentes páginas.

Línea de flujo. Indica en el sentido en el que se van a realizar las operaciones.

Línea conectora. Se usa para unir dos objetos.

Llamada a subrutina. Se usa para hacer una llamada a un subalgoritmo.

2.5.4.- Diagrama de Nassi-Schneiderman

- **Combinación de pseudocódigo y diagrama de flujo**
- **Las instrucciones se sitúan en rectángulos**
- **Poco usado**

Nombre del algoritmo
Acción 1
Acción 2
. . .
Acción n
Fin_algoritmo

2.6.- Fundamentos del lenguaje C

2.6.1 Historia

1967 – Martin Richards inventa el lenguaje BCPL

1969 – Aparece el S.O. UNIX (Ritchie y Thompson)

1970 - Aparece el lenguaje B escrito por Ken Thompson para el primer sistema UNIX de la DEC PDP-7, basado en BCPL.

1972 – Dennis M. Ritchie escribe el lenguaje C y se reescribe el S.O. UNIX en C. Inicialmente muy ligado a UNIX, posteriormente se reescriben compiladores para todas las plataformas.

1978 - Ritchie y Brian Kernighan publican la primera edición de “El lenguaje de programación C”, también conocido como La biblia de C.

1989 – El organismo americano de estándares, ANSI normaliza el lenguaje.

1990 – El organismo internacional de estandarización ISO, redacta otro documento de estandarización.

2.6.2 Características principales

- Es un lenguaje de **propósito general.** Inicialmente orientado a la implementación de sistemas operativos
- Es un lenguaje de **nivel medio** ya que dispone de estructuras de alto nivel pero a la vez permite el control a bajo nivel.
- Es un lenguaje **estructurado**, ofrece un control de flujo franco y lineal, condiciones, ciclos, anidamiento y subprogramas. En C todo se hace mediante la definición de funciones. Permite un diseño modular
- Falta de restricciones en **combinaciones de tipos** de datos
- No está ligado a ningún hardware ni arquitectura concretos. Esto lo hace un lenguaje **muy portable**, capaz de escribir programas que funcionen sin cambios en una gran variedad de máquinas.

2.6.2 Características principales

- Es un lenguaje **pequeño**: tiene muy pocas instrucciones propias y sólo 32 palabras reservadas. El resto de mecanismos de alto nivel lo proporcionan funciones especificas que se encuentran en bibliotecas del C.

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

- Es un lenguaje **eficiente**. Cualquier compilador de C es capaz de generar un eficiente programa ejecutable desde un fichero fuente .c . Un programa ejecutable en C tiende a ser eficiente en espacio y tiempo.
- C mantiene la filosofía de que los programadores saben lo que están haciendo, por tanto es muy **ineficiente** detectando errores.

2.6.3 Elementos del lenguaje

- **Comentarios** /* Comentario de C*/ //Comentario de C++
- **Identificadores:**
  - Longitud máxima de 31 caracteres
  - Se distingue minúsculas y mayúsculas
  - Debe ser diferente a cualquier palabra clave o nombre de función de biblioteca
  - Formado por los caracteres del alfabeto ingles, los dígitos del 0 al 9 y el carácter de subrayado,
  - Debe comenzar siempre por una letra.

2.6.3 Elementos del lenguaje

- **Tipos de datos básicos o atómicos**:

Tipo	Tamaño de bits	Rango
char	8	-128 a 127
int	16	–32768 a 32767
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308
void	0	Sin valor

- **Modificadores de tipos de datos**:
  - **char**: signed, unsigned
  - **int**: signed, unsigned, long y short
  - **double**: long.

2.6.- Fundamentos del lenguaje C

Identificador de tipo	Significado	Rango de valores
char	Carácter	-128 a 127
int	Entero	-32.768 a 32.767
short	Entero corto	-32.768 a 32.767
long	Entero largo	-2.147.483.648 a 2.147.483.647
unsigned char	Carácter sin signo	0 a 255
unsigned	Entero sin signo	0 a 65535
unsigned short	Entero corto sin signo	0 a 65.535
unsigned long	Entero largo sin signo	0 a 4.294.967.295
float	Real (coma flotante)	3,4E+/-38(7dígitos)
double	Real doble precisión	1,7E+/-308(15 dígitos)
long double	Real doble prec largo	1,7E+/-4932(15 dígitos)

2.6.3 Elementos del lenguaje

- **Variables**
  - Todas las variables deben ser declaradas antes de su uso.
  - Una declaración informa al compilador del nombre identificador asociado a cada variable así como de su tipo.
  - Se puede pensar en una variable como en una posición específica de memoria, reservada para un tipo específico de datos y con un nombre para referenciarla fácilmente.
  - Esencialmente se usan variables para permitir que la misma posición de memoria pueda tener diferentes valores del mismo tipo en instantes de tiempo distintos

2.6.3 Elementos del lenguaje

- **Variables**

La plantilla general de declaración de una variables es:

[Modificador_ almacenam.] [Modificador_ signo] [Modificador_ tamaño] tipo nombre;

Ejemplos:

```
int num;
int num = 2;
char c = ‘x’;
int n1,n2,n3;
```

2.6.3 Elementos del lenguaje

- **Constantes:**

Se pueden definir :

- Utilizando la directiva **define**:

  #define MAX 1000 (OJO! no lleva punto y coma al final de la instrucción ni operador =)

- Otra forma de definir constantes en C ISO es añadir a la declaración de una variable la directiva **const** delante

  const double e= 2.71828182845905;

2.6.3 Elementos del lenguaje

- **Constantes literales**
  - **Numéricas:**
    - **Enteros:** ejemplo: 1, - 2..
    - **Reales:** ejemplo: 2.3
  - **Caracteres:** ejemplo: ‘a’, ‘b’
  - **Cadenas de caracteres:** ejemplo: “esto es una cadena”

2.6.4.- Operaciones básicas y expresiones

- **Operadores:**
  - **De asignación**
    - El signo igual es el operador de asignación (=).
    - El operando de la izquierda debe ser una variable mientras que el de la derecha puede ser una constante, otra variable o bien una expresión.

Ejemplos:

```
int numero;

numero = 7;


char car= ‘m’;
```

2.6.4.- Operaciones básicas y expresiones

- **Operadores:**
  - **De asignación compuestos**

OPERADOR	SENTENCIA ABREVIADA	SENTENCIA NO ABREVIADA
=	=	=
++	m++	m=m+1
--	m--	m=m-1
+=	m+=n	m=m+n
-=	m-=n	m=m-n
*=	m*=n	m=m*n
/=	m/=n	m=m/n
%=	m%=n	m=m%n

2.6.4.- Operaciones básicas y expresiones

- **Operadores:**
  - **Monarios:**
    - incremento (++) y decremento (--)

      Pueden ir delante (preincremento) o detrás (postincremento)

      ¡Ojo cuando se usen dentro de una expresión!

      x=x+1; es equivalente a ++x; y a x++

      x=x-1 es equivalente a --x y a x –
    - **sizeof ():** Devuelve un entero que representa el tamaño en bytes del tipo indicado entre paréntesis.

      Ejemplo: sizeof(int) //Devuelve 2 si un entero ocupa 2 bytes

2.6.4.- Operaciones básicas y expresiones

- **Operadores:**
  - **Aritméticos**

Operador	Significado	Ejemplo
+	Suma	res = 3 + 5 (res ->8)
-	Resta	res = 5 – 3 (res ->2)
*	Multiplicación	res = 5 * 3 (res->15)
/	División entera	res = 10/ 3 (res ->3)
%	Módulo	res = 3 % 2 (res ->1)

  - Para realizar otras operaciones se recurre a las funciones de la librería “math.h”: sqrt(n), pow(base,exp)

2.6.4.- Operaciones básicas y expresiones

- **Operadores relacionales y lógicos:**
  - **Lógicos:**

OPERADOR	SIGNIFICADO
!	Not (NO lógico)
&&	And (Y lógico)
||	Or (O lógico)

  - **Relacionales**

OPERADOR	SIGNIFICADO
==	Igual a
!=	No igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

2.6.4.- Operaciones básicas y expresiones

- **Expresiones:**
  - En C el valor **0 es FALSO** y **cualquier valor distinto de cero es VERDADERO.**
  - Las expresiones construidas con operadores lógicos y/o relacionales al evaluarse devuelven un valor 1 o 0.
  - Es muy importante no confundir en una expresión el operador = con el operador ==, ya que su funcionamiento es completamente diferente.
  - Podemos combinar en una expresión operadores diferentes:

    Ejemplo: 10 > 5 && ! (10 < 9) || 4 < = 4 que es verdadero

2.6.4.- Operaciones básicas y expresiones

- **Expresiones:**
  - **Precedencia de operadores**

Operador	Asociatividad
() []	De izquierda a derecha
- ++ -- ! ~ * & sizeof(tipo)	De derecha a izquierda
* / %	De izquierda a derecha
+ -	De izquierda a derecha
<< >>	De izquierda a derecha
< <= > >=	De izquierda a derecha
== ¡=	De izquierda a derecha
&	De izquierda a derecha
&&	De izquierda a derecha
||	De izquierda a derecha
?:	De derecha a izquierda
= *= /= %= += -= &= <<= >>=	De derecha a izquierda
,	De izquierda a derecha

2.6.4.- Operaciones básicas y expresiones

- **Expresiones:**
  - **Conversión de tipos:**
    - **Ímplícita:** Se da cuando se mezclan expresiones aritméticas de un tipo con las de otro.
      - **Sentencia de asignación**: El valor del lado derecho se convierte al tipo de la variable del lado izquierdo pudiendo haber pérdida de información.

        Ejemplo: unaVariableEntera = unaVariableReal * 0.5
      - **Expresiones aritméticas**: si hay variables o valores de distintos tipos el compilador realiza determinadas conversiones antes de evaluar la expresión convirtiendo todo al tipo de mayor rango.

        Ejemplo: 3 + 5.3 → 3.0 + 5.3 = 8.3

2.6.4.- Operaciones básicas y expresiones

- **Expresiones:**
  - **Conversión de tipos:**
    - **Explícita (typecast):** El resultado de una expresión o una variable se puede cambiar explícitamente a otro tipo utilizando la notación (tipo) expresión o tipo (expresión)

      Por ejemplo :

      float f=0.5;

      f=f+1/2 == f=0.5+0 == 0.5

      f=f+float(1/2) == f=0.5+0.5 == 1

2.6.4.- Operaciones básicas y expresiones

- **Operaciones de entrada y salida**
  - Estas operaciones se encuentran en la biblioteca estándar de E/S ***stdio.h,*** por tanto debe incluirse en la cabecera del módulo o programa: #include<stdio.h>
  - **Entrada y salida de caracteres**
    - **int getchar (void):** No recibe ningún argumento, devuelve un entero. Lee un carácter del teclado dando un eco por pantalla.

      Ejemplo: c= getchar(); Almacenará en c el carácter que introduzcamos por teclado.
    - **int putchar (int c)** : Recibe un entero, devuelve también un entero. Escribe un carácter por pantalla.

      Ejemplo: putchar (c); Escribe el contenido de la variable entera c, cómo un carácter.

2.6.4.- Operaciones básicas y expresiones

- **Operaciones de entrada y salida (E/S)**
  - **Entrada y salida con formato**:
    - **int printf (cadena de caracteres, exp, variables o valores):**
      - Se usa para escribir información por pantalla con un formato determinado.
      - Sus argumentos son una cadena con formato y posiblemente una o varias expresiones, variables o valores de un tipo y número acorde con lo expresado en la cadena con formato.
      - La cadena de caracteres 'con formato': es una cadena entre comillas con
        1. Especificadores de formato precedidos de % (ver tabla) que debe corresponder con el número y tipo de los siguientes argumentos de la función.
        2. Caracteres directamente imprimibles.

2.6.4.- Operaciones básicas y expresiones

- **Operaciones de entrada y salida**
  - **Entrada y salida con formato**:
    - **int scanf (cadena de caracteres, variables):** Sus argumentos son una cadena 'con formato' y una o más variables (precedidas por &)
    - Se usa para leer información con formato desde teclado.
    - Sus argumentos son una cadena con formato y una o varias variables precedidas por el símbolo &.
    - La cadena de caracteres 'con formato': es una cadena entre comillas con:
      1. Especificadores de formato precedidos de % (ver tabla) que debe corresponder con el número y tipo de los siguientes argumentos de la función (las variables precedidas de &).

2.6.- Fundamentos del lenguaje C

- **Especificadores de formato**

Código	Formato
%c	Un único carácter
%d	Decimal
%nºd	Indica la longitud total del número
%ld	Entero largo (long)
%hd	Entero corto (short)
%i	Decimal
%e	Notación científica
%f	Decimal en punto flotante
%nº.nºf	Indica la longitud de la parte entera y la de la parte decimal
%.nºf	Indica la longitud de la parte decimal
%g	Usar %e o %f, el más corto
%o	Octal
%s	Cadena de caracteres
%u	Decimal sin signo
%x	Hexadecimales
%%	Imprime un signo %
%p	Muestra un puntero

- **Secuencias de escape**

Código	Significado
\a	alarma
\b	Espacio atrás
\f	Salto de página
\n	Salto de línea
\r	Retorno de Carro
\t	Tabulación Horizontal
\”	Comillas dobles
\’	Comilla simple
\0	Nulo
\\	Barra invertida
\v	Tabulación vertical
\xdd	Carácter ASCII cuyo código hexadecimal es *dd*
OOO	Carácter ASCII cuyo código octal es *OOO*

2.6.5.- Estructura general de un programa en C

- Las palabras clave del lenguaje tienen que ir en minúsculas. Por convenio, las constantes en mayúsculas.
- Inicio y fin de bloque: ‘{‘ ‘}’ respectivamente. En particular el bloque de main
- Siempre debe existir la función principal main: int main ()
- Al final de la función main, return (0);
- Para que el compilador reconozca funciones invocadas, es necesario indicar la librería (fichero con extensión.h) en la que residen. #include <libreria.h>
- Todas las sentencias acaban en ;

2.6.5.- Estructura general de un programa en C

```
/*Directivas del preprocesador*/

#include ...................

#define ..........

/*Declaración de variables globales y externas*/

/*Definiciones de tipos*/

/*Declaración de funciones*/

/*Implementación de funciones incluida la función main:*/

int main()
{
    /*Declaración de variables locales a main*/
    /*Sentencias del programa*/
    return 0;
}
```

2.7 Ejemplo

Siguiendo los pasos para la realización de un programa y utilizando todos los métodos de descripción de algoritmos estudiados se desea resolver el siguiente problema:

Realiza un programa que dado un valor para el radio escriba la longitud de la circunferencia, el área del círculo y el volumen de la esfera.

2.7 Ejemplo

1.- Planteamiento y análisis del problema

- Definición: ***programa que dado un valor para el radio escriba la longitud de la circunferencia, el área del círculo y el volumen de la esfera.***

- Representación de los datos
  - Entradas
    - Variables: Radio (Tipo real)
    - Constantes: PI 3.1416
  - Salidas
    - Variables: Longitud, Área y Volumen (Tipo real)
  - Recursos:
    - Longitud= 2ΠRadio
    - Área= ΠRadio^2
    - Volumen: 4/3ΠRadio^3

2.7 Ejemplo

2.- Diseño del Algoritmo

- Descripción en Lenguaje Natural

1) **Inicio.**
2) **Leer Radio.**
3) **Calcular Longitud, Superficie y Volumen**
4) **Escribir resultados.**
5) **Fin.**

- Descomposición en subproblemas
- Diseño descendente
- Refinamientos sucesivos
- Optimización y Depuración

2.7 Ejemplo

2.- Diseño del Algoritmo

- Descripción en Lenguaje Natural

1) **Inicio.**
2) **Leer Radio.**
3) **Calcular Longitud**
4) **Calcular Superficie**
5) **Calcular Volumen**
6) **Escribir Longitud, Superficie y Volumen.**
7) **Fin.**

2.7 Ejemplo

2.- Diseño del Algoritmo

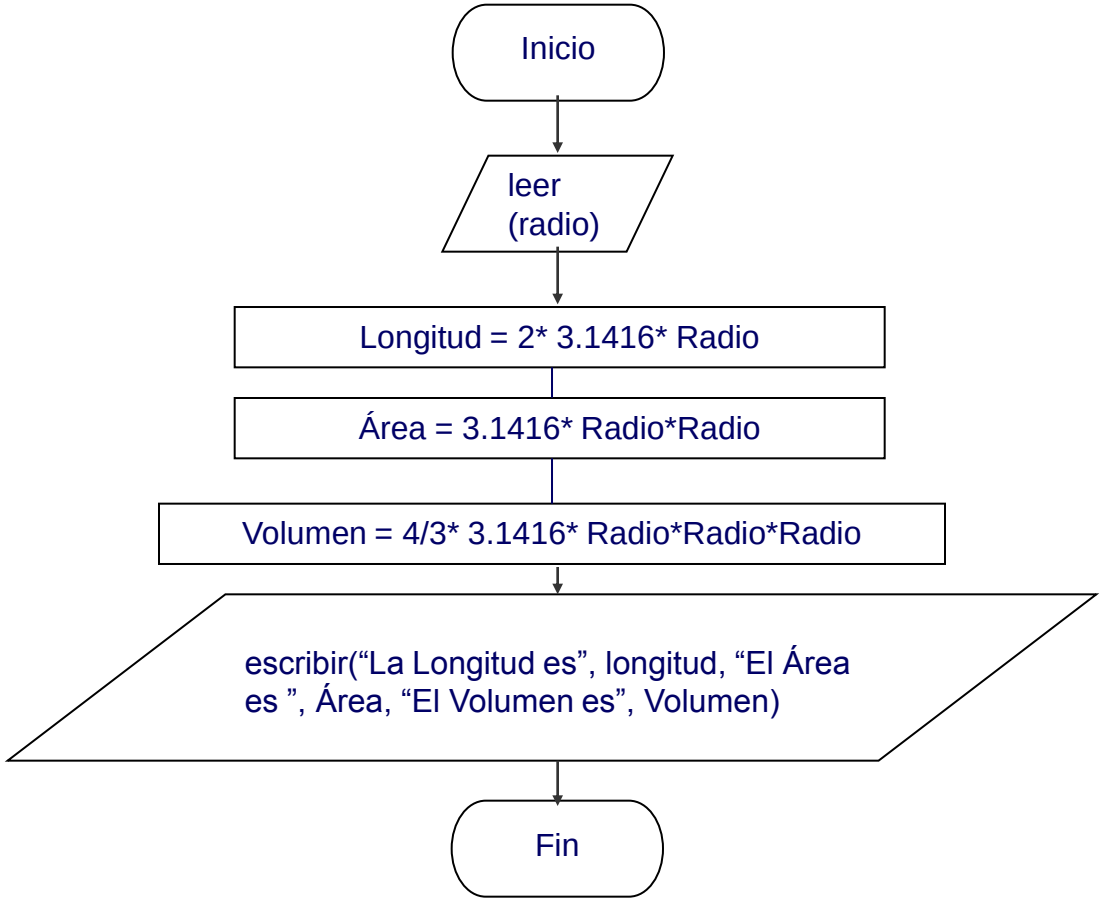
- Descripción en Lenguaje Natural

1) **Inicio.**
2) **Leer Radio.**
3) **Longitud= 2*3.1416*Radio**
4) **Superficie = 3.1416*(Radio*Radio)**
5) **Volumen= 4/3*3.1416*(Radio*Radio*Radio)**
6) **Escribir Longitud, Superficie y Volumen.**
7) **Fin.**

2.7 Ejemplo

2.- Diseño del Algoritmo

- Descripción en Diagrama de Flujo



2.7 Ejemplo

2.- Diseño del Algoritmo

- Descripción en Diagrama de **Nassi-Schneiderman**

Inicio
Leer (radio)
Longitud = 2* 3.1416* Radio
Área = 3.1416* Radio*Radio
Volumen = 4/3* 3.1416* Radio*Radio*Radio
escribir(“La Longitud es”, longitud, “El Área es ”, Área, “El Volumen es”, Volumen
Fin

2.7 Ejemplo

2.- Diseño del Algoritmo

- Descripción en Pseudocódigo

```
//Algoritmo que calcula la longitud de una circunferencia,
//el área del círculo y el volumen de la esfera.
Algoritmo Cálculos
const
    PI= 3.1416
var
    real: radio, longitud, area, volumen
inicio
     leer(radio)
     longitud ← 2* PI*radio
    area ← 2* PI* (radio* radio)
    volumen← (4div3) * PI *(radio*radio*radio)
     escribir("La longitud de la circunferencia es igual a", longitud, "El área
    del círculo es", area, "El volumen de la esfera es", volumen)
fin_algoritmo
```

2.7 Ejemplo

3.- Implementación del Algoritmo

- Lenguaje C

```
/*Algoritmo que calcula la longitud de una circunferencia,
el área del círculo y el volumen de la esfera.*/
#include <stdio.h>
#define PI 3.1416
int main()
{
   float radio, longitud, area, volumen;
   printf("Introduzca el radio");
   scanf("%f",&radio);
   longitud = 2* PI*radio;
   area = 2* PI* (radio* radio);
   volumen= (4/3) * PI *(radio*radio*radio);
   printf("La longitud de la circunferencia es igual a %f, El área del círculo es
     %f y El volumen de la esfera es %f", longitud, area, volumen);
}
```