

## Introducción a la Programación con C



Edificio de la división de Matemáticas e  
Ingeniería  
Oficina 10.



Universidad Nacional Autónoma de México  
Facultad de Estudios Superiores Acatlán  
Grupo de Cómputo Cuántico y Científico



Proyecto PAPIME PE104919

# Tabla de Contenido

<b>Licencia</b>	<b>iv</b>
<b>I Introducción</b>	<b>vi</b>
<b>Generalidades</b>	<b>vii</b>
1. Criterios generales de evaluación . . . . .	viii
2. Sobre los exámenes de primera y segunda vuelta . . . . .	ix
3. Actividades . . . . .	ix
<b>II Introducción: Hardware y Software</b>	<b>1</b>
<b>1. La computadora</b>	<b>1</b>
1. Antecedentes del software . . . . .	2
1.1. Definición de Software . . . . .	4
1.2. El Sistema Operativo . . . . .	4
<b>2. Sistemas de numeración</b>	<b>5</b>
1. Sistema binario, hexadecimal y octal . . . . .	6
1.1. Sistema binario . . . . .	6
1.2. Operaciones en sistema binario . . . . .	7
1.3. Conversión del sistema decimal al sistema binario . . . . .	8
1.4. El sistema hexadecimal . . . . .	8
2. Conversiones entre sistemas binario, octal, hexadecimal . . . . .	9
<b>III Herramientas para la solución de problemas a través de la Computadora</b>	<b>11</b>
<b>3. Solución de problemas</b>	<b>11</b>
1. Fases en la solución de problemas . . . . .	11
1.1. Análisis del problema . . . . .	11
1.2. Diseño . . . . .	11
•. Herramientas de programación . . . . .	12
1.3. Resolución a través de la computadora: Codificación (programa) . . . . .	14
•. Compilación y ejecución . . . . .	14
1.4. Paradigmas de Programación . . . . .	15
2. Repaso . . . . .	16
2.1. Pseudocódigo . . . . .	16
2.2. Python . . . . .	16
2.3. Diagramas de flujo . . . . .	16
<b>IV Introducción a la programación en C</b>	<b>18</b>
<b>4. La programación</b>	<b>18</b>
1. Lenguajes de programación . . . . .	18
1.1. Definiciones (programación, programa, lenguaje) . . . . .	18
1.2. Traductores del lenguaje (compiladores e intérpretes) . . . . .	19
2. Algunos conceptos de programación . . . . .	19
2.1. Identificadores . . . . .	19
2.2. Secuencias de escape . . . . .	20
2.3. Los formatos . . . . .	20
2.4. Operadores . . . . .	20
•. Operadores de asignación y unarios . . . . .	20

• Operadores relacionales . . . . .	21
• Operadores lógicos (booleanos) . . . . .	21
• Operadores especiales . . . . .	22
• Operadores de direcciones . . . . .	22
2.5. Algunos comentarios generales . . . . .	22
3. Pasos para crear un programa . . . . .	23
<b>5. Tipos de datos . . . . .</b>	<b>23</b>
1. Representación básicos . . . . .	23
2. Propiedades de cada objeto en C . . . . .	24
3. Representación de enteros . . . . .	25
4. Constantes simbólicas . . . . .	27
<b>6. El lenguaje de programación . . . . .</b>	<b>28</b>
1. Conversiones . . . . .	28
2. Sentencias y operadores . . . . .	29
3. Control de bucles . . . . .	32
4. Funciones de usuario . . . . .	32
4.1. Apuntadores . . . . .	34
4.2. Pasos de parámetros por referencia . . . . .	34
4.3. Algunos usos de los apuntadores . . . . .	35
4.4. Apuntadores nulos . . . . .	35
4.5. Funciones en línea, macros con argumentos . . . . .	36
5. Funciones de biblioteca . . . . .	36
• Funciones numéricas . . . . .	37
6. Tipos de almacenamiento . . . . .	37
• Tipo de almacenamiento: auto . . . . .	37
• Tipo de almacenamiento: static . . . . .	38
• Tipo de almacenamiento: extern . . . . .	38
• Tipo de almacenamiento: register . . . . .	39
• Ejemplos de almacenamiento . . . . .	40
7. Arreglos uni y multidimensionales . . . . .	40
7.1. ¿Cómo se escriben arreglos en C? . . . . .	41
7.2. Arreglos bidimensionales . . . . .	43
8. Concatenación . . . . .	43
9. Creación de archivos . . . . .	44
10. Estructuras . . . . .	45
11. Análisis de tiempos en un código . . . . .	45
11.1. Estructuras de control . . . . .	46
11.2. Teorema de Böhm y Jacopini . . . . .	47

## Licencia

Este documento está creado con fines educativos. La información contenida está sometida a cambios y a revisiones constantes, por lo que se sugiere no imprimirlo.

Puedes compartir el documento con quien desees; sin embargo debes respetar la autoría original. Puedes citar el material y considerarlo como referencias en tus proyectos.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

Esta obra está bajo una licencia **Creative Commons** “Reconocimiento-NoCommercial-CompartirIgual 3.0 España”.



Para citar este documento:

### 1. Bibtex

```
@manual{introCppGCCyC,  
  title  = "{Introducci\`on a la Programaci\`on con C}",  
  author = "{Orduz-Ducuara, J.A.}",  
  organization = "{Grupo de C\`omputo Cu\`antico y Cient\`ifico}",  
  month  = "{Febrero}",  
  year   = "{2020}",  
  address = "{FESAC-UNAM}",  
  note = "Los miembros del GCCyC aporta constantemente al desarrollo  
de este proyecto",  
}
```

### 2. bibitem

```
\bibitem{introCppGCCyC}  
Orduz-Ducuara, J.A.  
\textit{Introducci\`on a la Programaci\`on con C}.  
Los miembros del GCCyC aporta constantemente al desarrollo  
de este proyecto  
GCCyC, FESAC-UNAM, 2020.
```

Miembros activos del GCCyC:  
Salvador Uriel Aguirre Andrade  
Miguel Aguilar Hilario  
Miguel González Briones  
Alfonso Flores Zenteno  
Ana Karen Del Castillo  
Diana De Luna  
Leslie Valeria Vivas Laurrabaquio  
Zitlalli Nayeli Avilés Palacios  
Eledtih Andrea González Sánchez  
Edgar Ivan Martínez Villafañe  
Oscar Jair Vargas Palacios

Dr. Javier A. Orduz-Ducua <sup>a</sup>  
Profesor de Carrera Asociado C.  
Edificio de la división de Matemáticas e Ingeniería  
Oficina 10.  
FES Acatlán-UNAM

---

<sup>a</sup>Miembro que contribuyó al desarrollo de este material

## Parte I: Introducción

### Introducción general del curso

Aprender un lenguaje de programación es similar, en cierto modo, a aprender un nuevo idioma diferente al nativo. Un lenguaje de programación, se parece a un idioma, tiene su sintaxis (reglas del lenguaje), asimismo, en computación, existe el concepto de semántica: en pocas palabras la matemática detrás de la programación. Este conjunto de características son propias de cada lenguaje. Al final, un lenguaje de programación y un idioma son formas de expresión y de comunicación. Este aprendizaje puede implicar tanta pasión y compromiso como el discente quiera: la satisfacción por una nueva forma de expresión es propia de cada individuo. Después de incluir nuevas palabras, conceptos y reglas a la forma de expresión, seguramente, el lector se sentirá satisfecho y optará por aprender un nuevo lenguaje.

En estas notas se dejan las bases de la programación en C. Por cuestiones de fluidez, este documento omite la semántica de la programación y expone los principios de la sintaxis del lenguaje. El propósito es compartir el conocimiento con aquellos interesados en el autoaprendizaje y el desarrollo de habilidades computacionales, más que fortalecer impartir la lógica detrás de la programación. Sin embargo, es importante que el lector comprenda que la programación tiene profundos conceptos matemáticos que no se discuten en este trabajo.

En este manual, el lector encuentra ejercicios que puede mantener ocultos hasta que dé clic sobre la palabra "solución" para mostrar el resultado y comparar su respuesta. Se anima al lector que mantenga oculta la respuesta hasta proponer una solución. Además, el estudiante debe considerar que la respuesta puede tener diferentes soluciones. No se desanime.

Por otro lado, para mí (Javier Orduz) es un orgullo comentar que este trabajo ha sido desarrollado por los miembros del GCCyC (grupo que yo inicié), un grupo muy joven en la FESAc-UNAM, con miembros muy activos, que ven en sus compañeros: un equipo y un amigo de quien aprender; y con quien compartir. Esta generación aporta bastante al desarrollo de la institución y del país. Es un grupo que ha nacido por iniciativa de jóvenes inquietos que me dieron la oportunidad de acompañarlos en una parte de su etapa de formación y que me brindaron su confianza para pensar en un ejercicio académico como el Grupo de Cómputo Cuántico y Científico. Espero aportar al grupo y continuar con la recepción de las enseñanzas que me brindan cada día. Agradezco mucho su apoyo.

Es posible que algunos miembros que contribuyeron al inicio de este proyecto hayan sido ignorados, pero no olvidados, así que agradeceré que me contacten para dar el respectivo reconocimiento.

Además de la siguiente división del contenido se sugiere que el usuario revise los códigos que se dejan en Github, GitLab, Jupyter y otros repositorios que están asociados a este curso, y que pueden ser consultados desde el sitio: [yeyecoacatlan.unam.mx/](http://www.yeyecoacatlan.unam.mx/)<sup>1</sup>. El contenido de este documento es: capítulo 1, hay revisión de conceptos básicos sobre la computadora: software y hardware; capítulo 2, se exponen los sistemas de numeración, capítulo 3, se discuten los operadores las fases para solución de problemas a través de la computadora; capítulo 4, se introducen los temas de programación. Capítulo 5, se presentan los tipos de datos básicos de C. Finalmente (cap. 6), se exponen los temas básicos de programación y se realizan ejercicios.

---

<sup>1</sup>La dirección es <http://www.yeyecoacatlan.unam.mx/>

## Generalidades

- **G-2202** de 9 a 11 horas

Horario de la clase:

- Lunes (A-425),
- Miércoles (A-723) y
- viernes (A-425).

- **G-2251** de 14 a 16 horas

Horario de la clase:

- Lunes (A-422).
- Miércoles (A-422).
- viernes (A-723).

La hora máxima de llegada es  $t + 15$  minutos, donde  $t$  es la hora de inicio de clase.

### Fin de ciclo escolar 22 de mayo 2020

Días inhábiles y asueto académico:

- 03 de febrero, 2020
  - 16 de marzo, 2020
  - 01, 10, 15 de mayo, 2020
  - 06 al 10 de abril, 2020
- Más información sobre mi horario aparece en: <http://www.mac.acatlan.unam.mx/portada/profesores/0/>.
  - Además en el sitio: **SEA** (<http://sea.acatlan.unam.mx/>), encontrarán más información sobre el curso para matricularse deben usar: **u815Tgb**. Consultarlo para tareas, material, bibliografía, información y preguntas.
  - En el sitio <https://www.codechef.com/getting-started> encontrarán ejercicios y más material relacionado a la programación.
  - Les dejo un google classroom (código de clase **n2f37gz**)
  - Además se deben realizar los ejercicios que se dejan en [www.code.org](http://www.code.org) con el código: **YRFBNL**. Debes realizar el 70 % del curso para tener derecho a presentar examen y proyecto.
  - Tenemos un sitio en Github: <https://github.com/UNAM-FESAc/> y <http://www.yeyecoacatlan.unam.mx> donde encontrarán: algunos materiales e información (código en C) y otras cosas más. Además pueden revisar el repositorio del **curso anterior** (<http://www.yeyecoacatlan.unam.mx>) o en github (<https://github.com/UNAM-FESAc/>).

**Es posible que encuentren detalles por cambiar; agradeceré todos los comentarios y sugerencias.**

**Es deber de cada estudiante revisar constantemente el sitio SAE, google classroom, code.org, Github y los demás materiales que se comparten en clase.**

**Usaremos mucho material en inglés, así que los ánimo a que estudien, lean y se comuniquen en este idioma.**

- Todo el material (escrito) debe ser original, ¡cuidado con el **plagio**! Cuidado en los exámenes. Pueden revisar:
  - El código de ética de la UNAM: <http://www.ifc.unam.mx/pdf/codigo-etica-unam.pdf> y <http://eticaacademica.unam.mx/>
  - La legislación universitaria <https://goo.gl/M9G3cC> y defensoría de los derechos universitarios <http://tinyurl.com/v2jyr2s>

En la siguiente sección veremos algunos detalles sobre las calificaciones y los criterios de evaluación.


## 1. Criterios generales de evaluación

Estos son los criterios de evaluación. El alumno debe

- Comprender mucho material bibliográfico que estará en inglés, por lo tanto, es importante se capacite constantemente en este idioma.
- El material escrito debe estar presentado adecuadamente; es decir, limpio, escrito en computador y, además, debe respetar las normas de ortografía. Todos los documentos deben entregarse en documento **PDF** (excepto: algunos de mis *scripts* que no respetan acentos)
- Enviar los documentos finales (productos), tomando las sugerencias dadas: fechas y horas acordadas.
- Realizar y mantener ordenado el repositorio: Drobox, Drive, OneDrive, etc.. Ejemplos de repositorios:



Figura 1: Ejemplos de repositorios.

- Guardar el  en modo avión.
- Nota mínima aprobatoria **7.0**.

El porcentaje de la calificación total está dada por:



Primer examen (Semana del 16 al 20 de marzo):	30 %
Segundo examen (Semana del 04 de 08 de mayo):	30 %
Proyecto final (Semana del 11 de 15 de mayo):	20 %
Dropbox: Ejercicios y tareas (en o de laboratorio o en clase):	10 %
Material, información, tareas, ejercicios en SAE y en code.org	10 %

Respecto a la actitud y otros detalles

- Tomar una postura de responsabilidad y cumplimiento con la materia.
- Solamente el



- Es necesaria mucha comunicación.



Matengamos un ambiente de respeto, tolerancia y cordialidad.

## 2. Sobre los exámenes de primera y segunda vuelta

Estos exámenes se llevan a cabo durante la semana del 25-29 de mayo de 2020 (primera vuelta) y la semana del 01-05 de junio de 2020 (segunda vuelta). Estas fechas las envía el programa de MAC.

Estos exámenes se llevarán a cabo en las fechas acordadas por el programa de MAC, su evaluación máxima de  $6 \leq x_f \leq 8$  y se evaluará todo el material revisado en el curso.

## 3. Actividades

- GCCyC y Robocop . JAOD
- Seminario de investigación en CTIM. JAOD *et. al.*
- MAC-Day. Mtra. G. Eslava.

Tarea: Manifiesto. Programación II. 2020-II

A quien interese:

Por este medio, yo, \_\_\_\_\_ con matrícula \_\_\_\_\_, del grupo \_\_\_\_\_, de la carrera de \_\_\_\_\_, de la FES Acatlán, declaro que estoy enterado del temario de Programación II, fechas de inicio y fin del curso, y de exámenes; tiempos de llegada a cada clase, sitios web de consulta (Github, code, SEA, yeyeco, google classroom, entre otros); de los materiales, criterios y formas de calificar; y de los porcentajes de calificación a través de exámenes y proyecto. Además, conozco los requerimientos del curso: respeto, compromiso y disciplina. Por lo tanto firmo, de manera legible, coloco mi nombre y subo este documento en el sitio SEA.

\_\_\_\_\_  
Nombre

\_\_\_\_\_  
Firma

\_\_\_\_\_  
Fecha

Tarea: Manifiesto. Programación II. 2020-II

A quien interese:

Por este medio, yo, \_\_\_\_\_ con matrícula \_\_\_\_\_, del grupo \_\_\_\_\_, de la carrera de \_\_\_\_\_, de la FES Acatlán, declaro que estoy enterado del temario de Programación II, fechas de inicio y fin del curso, y de exámenes; tiempos de llegada a cada clase, sitios web de consulta (Github, code, SEA, yeyeco, google classroom, entre otros); de los materiales, criterios y formas de calificar; y de los porcentajes de calificación a través de exámenes y proyecto. Además, conozco los requerimientos del curso: respeto, compromiso y disciplina. Por lo tanto firmo, de manera legible, coloco mi nombre y subo este documento en el sitio SEA.

\_\_\_\_\_  
Nombre

\_\_\_\_\_  
Firma

\_\_\_\_\_  
Fecha

## Parte II: Introducción: Hardware y Software

### Capítulo 1

#### La computadora

##### Objetivo

Describir los componentes y funcionamiento de la computadora digital.



Titan (última imagen) es la computadora más potente del mundo. Memoria 700 TB y 17.590 billones de operaciones de cálculo por segundo (17.5 petaflops). El rendimiento de un ordenador actual se mueve en el centenar de gigaflops <sup>1</sup>.



<sup>1</sup>Ver más información <http://blogthinkbig.com/asi-titan-supercomputadora-mas-potente-mundo/> y <https://www.olcf.ornl.gov/titan/>.

## 1. Antecedentes del software

Para esta sección usaremos el libro Jones (2013).

El humano ha requerido almacenar y manipular gran cantidad de información para diferentes fines. Al principio requería mantener la información de sus riquezas: animales, plantas, empleados, etc.



Posteriormente, la información se hizo mayor y se amplió. Por ejemplo, cálculos más complicados en ciencias. Las herramientas para asistir con las decisiones lógicas fueron las últimas en desarrollarse.

### RAE 2017

Conjunto de programas, instrucciones y reglas para ejecutar ciertas tareas en una computadora u ordenador. Se puede usar programas informáticos o aplicaciones informáticas.

Hacia 1930 y 1939 fue el inicio del software. Ha sido necesario e implementado en diferentes áreas:





Actualmente, tenemos una de las máquinas más poderosas que el hombre haya creado: LHC ubicado en la frontera Franco-Suiza. Veamos alguna información tomada del sitio oficial: <https://home.cern/science/computing>.

#### Actividad en clase: Discusión en SEA

La siguiente actividad representa la opinión personal de cada estudiante y debe tratarse con respecto.

Realiza un debate con tus compañeros de equipo y discute en el foro del curso:

1. ¿Qué tipos de cálculos usamos cada día?
2. ¿Qué tipo de información o datos necesitamos guardar?
3. ¿Cuáles son los mejores métodos para retención a largo tiempo de la información?
4. ¿Cuáles métodos de análisis pueden ayudar a hacer elecciones complicadas o decisiones?
5. ¿Cuáles son los mejores métodos de comunicación de datos y conocimiento?
6. ¿Consideras que las ciencias naturales, la matemática y la computación están relacionadas?
7. ¿Cuáles habilidades como estudiante de MAC debes desarrollar?
8. Escribe una opinión personal sobre la computación en el LHC

Estas preguntas son importantes para la historia de software y las computadoras.

**Actividad SEA**

Revisa la actividad SEA sobre la computadora. Observa el video, revisa la página (numeralia) y discute en el foro.

**1.1. Definición de Software**

Es el conjunto de instrucciones secuenciales que le indican al hardware los procesos que debe realizar.

El software es el responsable del correcto funcionamiento de del hardware.

Tipos de software:

- de sistema: gestiona los dispositivos.
- de aplicaciones: Apoya al usuario en la realización de tareas.
- de desarrollo: todas aquellas herramientas de software que nos ayuda a desarrollar más software.

**Software Libre...**

Existe el software **libre**, **gratuito** y **de propietario**: el primero puede ser copiado, distribuido y modificado; el segundo quiere decir que no es de paga, pero puede no ser libre; y el tercero no permite ser estudiado, modificado, copiado, ni distribuido.

**1.2. El Sistema Operativo**

La primera versión de Windows 1 se presentó en 1985.

La primera versión de Linux se presentó en 1991.

La primera versión de MAC OS X 1978.

**Concepto: El sistema operativo (SO)**

Es un conjunto de programas y órdenes que controlan los procesos básicos de una computadora. El SO permite que el funcionamiento de otros programas, gestiona el software y hardware; además sirve de interfase entre el humano y el ordenador.

El SO se comunica con los dispositivos de entrada y salida (E/S, I/O), administra procesos, la memoria y se encarga de las comunicaciones. El SO trae utilidades para configurar (personalizar) el equipo. Abordar el tema de los sistemas operativos requiere de más tiempo ya que son un tema muy amplio.

**Actividad SEA**

Crear una wiki colaborativa sobre **la computadora en el 2019 y su futuro**. Pueden usar imágenes, videos cortos, podcasts, texto, diapositivas y todo material que enriquezca el wiki. Importante que colaboren en el grupo al que pertenecen, en caso de no respetar esta regla se restará medio punto el primer examen. El objetivo de esta actividad es motivar el trabajo colaborativo. El propósito es vincular al grupo de estudiantes para que desarrollen trabajo en equipo.

Factor	Prefijo	Símbolo
$10^{21}$	zetta	Z
$10^{18}$	exa	E
$10^{15}$	peta	P
$10^{12}$	tera	T
$10^9$	giga	G
$10^6$	mega	M
$10^3$	kilo	k(K)
$10^{-3}$	mili	m
$10^{-6}$	micro	$\mu$
$10^{-9}$	nano	n
$10^{-12}$	pico	p
$10^{-15}$	femto	f
$10^{-18}$	atto	a
$10^{-21}$	zepto	z

## Capítulo 2

### Sistemas de numeración

Para esta sección usaremos el libro (Lyttton, 2002, pág. 281). La notación científica, nos sirve para:

- **Ejemplo 1:**  $2.3 \times 10^{-5} = 0.000023$
- **Ejemplo 2:**  $1700 \text{ g} = 1.7 \times 10^3 \text{ g} = 1.7 \text{ kg} = 1.700 \times 10^3 \text{ g}$

En los lenguajes de programación (por ejemplo, Fortran) se usa: 2.3E-5 o 2.3e-5, donde e  $\neq$  al símbolo que representa la base de los logaritmos naturales.

Es importante notar que hay cierto abuso del lenguaje, por ejemplo:  $1 \text{ k} = 1024 = 2^{10}$ . Un megabyte (MB) es  $1024^2$  bytes. Esto se usa para bits o bytes medidos directamente, pero no para razones de transferencia de información. Veamos el siguiente ejemplo.

**Ejemplo 0.1.** 12 kbits por segundo (bps), debería ser  $\frac{12000}{1024} \sim 11,7 \text{ kb}$  en 1 segundo. Importante tener en cuenta que 12kbps. Pensando en un CD que tiene 700 MB, realizando un cálculo tenemos:  $700000 \times 1024 = 716800000$  bytes, que se debería escribir como  $\frac{716800000}{1024^2} = \frac{716800000}{2^{20}} = \frac{716800000}{1048576} \sim 684 \text{ MB}$ .

Algunos símbolos que deberás tener presente:

phenomenon	abbrev.	unit	unit abbrev.
computer memory	RAM	byte	B
data		bit	b
charge	Q	coulomb	C
capacitance	C	farad	F
current	I or i	ampere, amp	A
voltage, potential	V or E	volt	V
energy	E	joule	J
volume	V	milliliter, ...	ml, cc
area	A	squared length	$\mu\text{m}^2$ , $\text{cm}^2$
velocity	v	meter per second	m/s
resistance	R	ohm	$\Omega$
conductance	g	siemens, mho	S
time	t	second	s
frequency	f	hertz	Hz, /s
period	T	second	s
temperature	T	degree	$^{\circ}\text{C}$ , $^{\circ}\text{K}$
diameter, length	d, L	micron	$\mu\text{m}$
chemical amount	n	mole	mol
concentration	[x]	molarity	M

Figura 2.1: Tomada de [Lytton \(2002\)](#)

Algunas conversiones:

1. Convierta 50 millones de milímetros en millas, recuerde que 1 pulgada = 2.54 cm, 1 pie = 12 pulgadas y 1 milla = 5280 pies.
2. Si viajas a 60 mph, ¿cuánto tiempo te toma viajar 1 pulgada?

Además, es importante tener en cuenta el análisis dimensional que nos ayudará a corroborar nuestros resultados.

## 1. Sistema binario, hexadecimal y octal

Los sistemas numéricos (o de numeración) que revisaremos son tres: Binario, Hexadecimal y Octal. Empezaremos con el sistema binario, cuya base es dos. El sistema binario, tiene dos símbolos (0, 1):

### 1.1. Sistema binario

Supongamos que tenemos los siguientes elementos:

| | | | | | | | | |

¿Qué pasa si encerramos los elementos de dos en dos, empezando desde la izquierda? ¿Podremos llegar a una nueva forma de representación? Intenta representarlo como una cadena de ceros y unos. Interpreta el resultado.



## 1.2. Operaciones en sistema binario

Es importante tener en cuenta la siguiente tabla

$\pm$	0	1
0	0	1
1	1	0

Para la multiplicación se debe considerar la siguiente tabla:

$\times$	0	1
0	0	0
1	0	1

Es importante considerar que:

1. En la resta binaria  $0 - 1 = 1$ , donde **1** es un acarreo, que evita (o contiene) el número binario negativo.
2. En la suma binaria  $1 + 1 = 1$  0, donde **1** es el acarreo obtenido de la suma binaria.

Veamos algunos ejemplos:

$$\begin{array}{r}
 \phantom{+} 1 \\
 + \phantom{1} 1 \\
 \hline
 1 \phantom{0}
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{+} 1 \phantom{0} \\
 + \phantom{1} 1 \\
 \hline
 1 \phantom{1}
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{+} 1 \phantom{0} \\
 + \phantom{1} 0 \\
 \hline
 1 \phantom{0}
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{+} 1 \phantom{0} \\
 - \phantom{1} 1 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{+} 1 \phantom{1} \phantom{1} \phantom{1} \\
 + \phantom{1} 1 \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 1 \phantom{1} \phantom{0} \phantom{1} \phantom{0}
 \end{array}$$

Ahora puras restas:

$$\begin{array}{r}
 \phantom{-} 1 \phantom{1} \\
 - \phantom{1} 1 \\
 \hline
 0
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{-} 1 \phantom{0} \phantom{0} \\
 - \phantom{1} 1 \phantom{1} \\
 \hline
 1 \phantom{1}
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{-} 1 \phantom{1} \phantom{1} \\
 - \phantom{1} 1 \\
 \hline
 1 \phantom{0}
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{-} 1 \phantom{1} \phantom{0} \\
 - \phantom{1} 1 \phantom{1} \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{-} 1 \phantom{1} \phantom{1} \phantom{1} \phantom{1} \\
 - \phantom{1} 1 \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \hline
 1 \phantom{1} \phantom{0} \phantom{1} \phantom{0}
 \end{array}$$

Ahora realiza las siguientes operaciones cuando (en el sistema decimal) el valor absoluto del sustraendo es mayor que el valor absoluto del minuendo.

$$\begin{array}{r}
 \phantom{-} 0 \phantom{1} \phantom{1} \phantom{0} \phantom{1} \\
 - \phantom{1} 1 \phantom{0} \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 \textcolor{red}{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \phantom{0}
 \end{array}
 \quad
 \begin{array}{r}
 \phantom{-} 0 \phantom{0} \phantom{1} \phantom{0} \phantom{1} \\
 - \phantom{1} 1 \phantom{1} \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 \textcolor{red}{1} \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{0}
 \end{array}$$

Donde **1** es el bit de signo.

Para realizar las anteriores operaciones es necesario obtener el complemento A2 del número binario.

**Ejemplo 1.1.** Supongamos que tenemos el número 15 en el sistema decimal, que equivale a 1 0 0 1 1 en el sistema binario.

**Formato Número-signo:**

**1** 1 0 0 1 1.

**Formato Complemento A1:**

**1** 0 1 1 0 0.

**Formato Complemento A2:**

**1** 0 1 1 0 1.

Observa que en el formato Número-signo introducimos un **1** que indica el signo del número (1 indica el signo negativo; y 0 indica positivo), en el formato complemento A1 se cambian todos los bits por su inverso, es decir 1 por 0 y 0 por 1. El complemento A2 se cambian todos los bits por su inverso, a partir del primer bit, de derecha a izquierda, cuyo valor es 1.

El resultado se obtiene calculando el complemento A2 del minuendo y aplicando la suma. Se debe mantener el primer dígito que indica el signo y al final se omite para obtener el resultado.

**Ejercicio 1.1.** Realiza las siguientes operaciones en clase:

$$\begin{array}{r} 1\ 0 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 1 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 0\ 0 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 0\ 1 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 1\ 0 \\ +\ 1 \\ \hline \end{array}$$

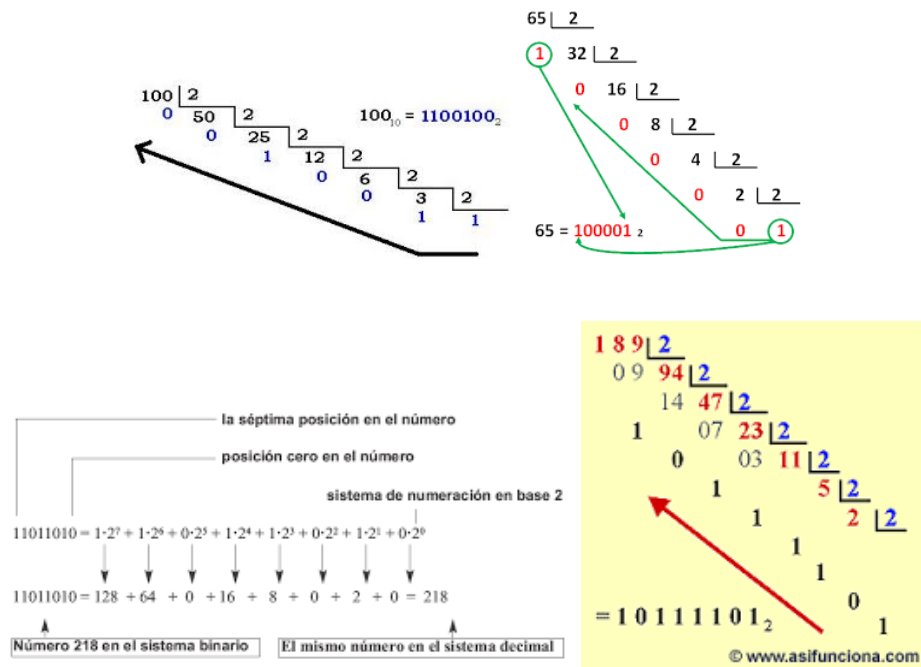
$$\begin{array}{r} 1\ 1\ 1 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 0\ 0 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 0\ 1\ 0 \\ +\ 1 \\ \hline \end{array} \quad \begin{array}{r} 1\ 0\ 1\ 1 \\ +\ 1 \\ \hline \end{array}$$

**Ejercicio 1.2.** Compruebe las anteriores operaciones en el sistema decimal.

$$\begin{array}{r} 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\ +\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1 \end{array} \quad \begin{array}{r} 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\ +\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1 \end{array}$$

### 1.3. Conversión del sistema decimal al sistema binario

Usaremos el siguiente método:



### 1.4. El sistema hexadecimal

El sistema hexadecimal, tiene 16 símbolos (básicos)

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	\$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

El sistema octal tiene 8 símbolos: 0,1,2,3,4,5,6,7. Conviene trabajar con este sistema en vez del sistema decimal, ya que es fácil de traducir al sistema binario.

## 2. Conversiones entre sistemas binario, octal, hexadecimal

**Ejemplo 2.1.** Convertir del sistema decimal fraccionario a octal

¿Cómo transformar un Decimal Fraccionario a Octal?

Convertir **323.625** a Octal

323 | 8

3 40 8

0 0 5 8

5 0

.625 x 8 = 5

5 0 3 . 5

**Ejemplo 2.2.** Convertir del sistema decimal al hexadecimal

Transformar = 250.25

250 | 16

10 15

0,25 X 16 = 4,00

A F 4

Resultado= FA.4

**Ejemplo 2.3.** Expresar  $257_{10} \rightarrow$  en el sistema decimal como una suma y producto en base 10.  $257_{10} = 2 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0$ .

**Ejemplo 2.4.** Expresar  $257_8 \rightarrow$  en el sistema decimal.  $257_8 = 2 \cdot 8^2 + 5 \cdot 8^1 + 7 \cdot 8^0$ . Ahora calcule su equivalente en el sistema decimal, obtenemos:  $175_{10}$ .

**Ejemplo 2.5.** *Expresé  $1001,101_2 \rightarrow$  en el sistema decimal.  $1001,101_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$ . El sistema decimal, obtenemos: 9.625.*

Octal to binary and hex to binary conversion						
octal digit	binary value		hex digit	binary value	hex digit	binary value
0	000		0	0000	8	1000
1	001		1	0001	9	1001
2	010		2	0010	A	1010
3	011		3	0011	B	1011
4	100		4	0100	C	1100
5	101		5	0101	D	1101
6	110		6	0110	E	1110
7	111		7	0111	F	1111

Figura 2.2: Tomada de [Lytton \(2002\)](#).

Para pasar de hexadecimal a binario solamente tenemos que reemplazar los caracteres en hexadecimal por sus equivalentes en binarios.

**Ejemplo 2.6.** *Expresemos de hexadecimal a binario*

$$AF_{16} = 10101111_2$$

**Ejemplo 2.7.** *Expresemos de hexadecimal a decimal*

$$AF_{16} = 10 \cdot 16^1 + 15 \cdot 16^0 = 175_{10}$$

**Ejemplo 2.8.** *Expresemos de hexadecimal a decimal*

$$AF_{16} = 10 \cdot 16^1 + 15 \cdot 16^0 = 175_{10}$$

## Parte III: Herramientas para la solución de problemas a través de la Computadora

### Capítulo 3

#### Solución de problemas

##### Objetivo

Describir los pasos para la solución de problemas utilizando técnicas de representación de algoritmos.

#### 1. Fases en la solución de problemas



Figura 3.1: Análisis del problema. Tomado de L. y Zahonero Martínez (2001)

##### 1.1. Análisis del problema

Esta fase requiere una definición clara y debe contemplar lo qué debe hacer el programa y el resultado o solución deseada.

Conviene tener en cuenta las siguientes preguntas:

- ¿Qué entradas requiere? (tipo y cantidad)
- ¿Cuál es la salida deseada? (tipo y cantidad)
- ¿Qué método produce la salida deseada?

En esta fase buscamos el qué vamos a hacer, esto implica análisis. En la siguiente fase debes responder otra pregunta.

##### 1.2. Diseño

Determinar cómo: *divide y vencerás*.

Lo ideal es usar un diseño descendente (*top-down*) o modular. Un **módulo** es un subprograma que tiene solamente una entrada y una salida.

Un módulo puede ser planeado, codificado, comprobado y depurado independientemente y luego combinarlos entre sí.

Lo anterior requiere:

- Programar el módulo
- Comprobar el módulo
- Depurar el módulo (si es necesario)
- Combinar el módulo con los demás.

El proceso que convierte los resultados del análisis del problema en un diseño modular con refinamientos sucesivos que permitan una posterior traducción a un lenguaje se denomina diseño del algoritmo.

El **refinamiento sucesivo** es el proceso de romper el problema y expresar cada paso de forma detallada.

La programación modular es el método de usado para romper el programa en módulos más pequeños.

Nótese la importancia que tiene el diseño sobre la práctica de la programación. Esto nos ayudará a solucionar problemas eficiente y nos reduce errores; para esto usaremos algunas herramientas. Las veremos en la siguiente sección.

#### ● Herramientas de programación

Para iniciar con el uso de las herramientas, debemos compartir algunos conceptos e ideas.

- El concepto de algoritmo

Un algoritmo es una lista bien definida, ordenada y finita de operaciones, que permite encontrar la solución a un problema determinado [Juganaru Mathieu \(2012\)](#).

- Descripción de algoritmos

##### 1. Descripción de alto nivel

- descripción del problema
- selección del modelo matemático
- explicación verbal del algoritmo (omitir detalles).

##### 2. Descripción formal

- Usar pseudocódigo o diagrama de flujo para describir la secuencia que conduce a la solución.

Los diagramas de flujo y pseudocódigos son herramientas muy usadas para diseñar algoritmos. [L. y Zahonero Martínez \(2001\)](#).

##### 3. Implementación

- Mostrar el archivo fuente del algoritmo.

Veamos algunos ejemplos de pseudocódigo y diagramas de flujo y su construcción.

#### Pseudocódigo

Es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectura de programas.

**Ejemplo 1.1.**

```

if estudio, trabajo con compromiso;
y soy disciplinado;
then
    tengo éxito;
    Logro mis objetivos;
    y tengo lana
else
    Tengo poca probabilidad de éxito.

```

**Algorithm 1:** Pseudo-código para if**Ejemplo 1.2.**

```

while te apetezca y tengas tiempo do
    ven a verme
end

```

**Algorithm 2:** Pseudo-código para mientras**Ejemplo 1.3.**

```

while no termine este documento do
    lea cada línea;
    if se entiende then
        vaya a la siguiente sección;
        y repase mentalmente los conceptos e ideas;
    else
        vaya a al inicio de la sección actual;
    end
end

```

**Algorithm 3:** Pseudo-código para mientras y si**Ejemplo 1.4.**

```

for i < 2 to 5 do
    algo;
end

```

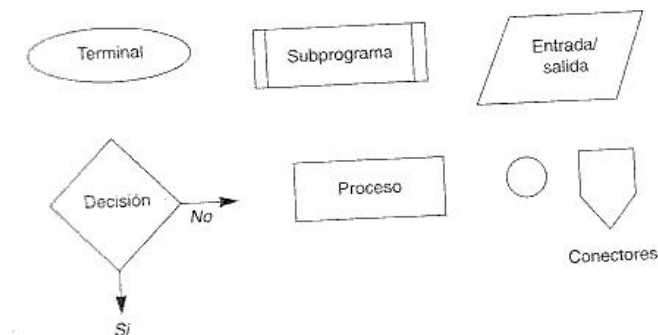
**Algorithm 4:** Pseudo-código para para

Otras herramientas usadas para la resolución de problemas son los diagramas de flujo, los veremos en la siguiente sección.

**Diagramas de flujo**

Es una representación gráfica de un algoritmo.

Puedes usar algunas herramientas para diseñar los diagramas flujo, por ejemplo: [LucidChart](#).



Algunos símbolos que se usan en diagramas de flujo están estandarizados por [ANSI](#) y la [ISO](#) se muestran en la figura 3.2.

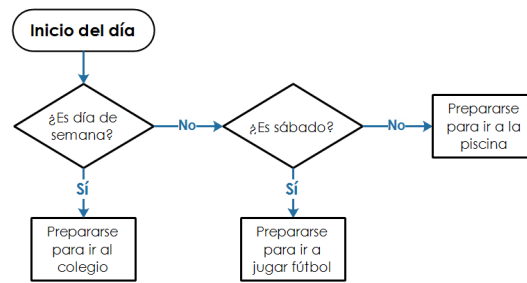


Figura 3.2: Elementos usados en los diagramas de flujo.

### 1.3. Resolución a través de la computadora: Codificación (programa)

Esta fase es la escritura de en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes.

Es la escritura en un lenguaje de programación de la representación del algoritmo. El diseño del algoritmo debe ser independiente del lenguaje de programación.

La conversión del algoritmo en programa de realizarse de la siguiente manera:

- Sustituir las palabras reservadas en español por sus homónimos en inglés y
- las operaciones (instrucciones) indicadas que están en el lenguaje natural deben expresarse en el lenguaje de programación.

#### ● Compilación y ejecución

El **programa o archivo fuente** contiene el programa que está almacenado en algún lugar del equipo. Este archivo deber ser traducido a **lenguaje de máquina**, este proceso se realiza con el compilador y el sistema operativo que se encarga de la compilación.

El programa se escribe en un editor de texto. A estas líneas se le llaman archivo fuente, que se traducen al lenguaje de máquina, este proceso se realiza con el compilador.

En caso de errores se necesita depurar y compilar nuevamente; una vez todo ha sido corregido se obtiene el **programa objeto** (no ejecutable) y, ahora, el SO realiza la fase de montaje (enlace) del programa objeto con las librerías del programa del compilador; este proceso de montaje genera un **programa ejecutable**.

#### Verificación y depuración

La verificación o compilación de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados datos de prueba. Esto determina si el programa tiene *bugs* (errores).

La depuración es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores.

Hay tre tipos de errores

- de compilación: por lo general son errores de sintáxis
- de ejecución: se producen por instrucciones que la computadora puede comprender pero ejecutar, e.g.: división por cero, etc.
- lógicos: está relacionado al diseño del algoritmo.



### Documentación y mantenimiento

Hay dos tipos de documentación que describe los pasos a seguir en el proceso de resolución de un problema: interna y externa:

La documentación interna es la contenida en las líneas de comentarios; la externa es aquella que ayuda al diseño del algoritmo y otros análisis previos a la implementación del programa.

### 1.4. Paradigmas de Programación

Un **Paradigma de programación** provee la visión y métodos de un programador en la construcción de un programa o subprograma [Juganaru Mathieu \(2012\)](#).

A continuación, enunciaremos algunos paradigmas de programación:

#### Paradigma Imperativo (PI):

En este paradigma se impone que cualquier programa es una secuencia de instrucciones o comandos que se ejecutan siguiendo una orde de arriba hacia abajo; este enlace unicamente se interrumpe para ejecutar otros subprogramas o funciones y, después, regresa al punto de interrupción. Los lenguajes de programación son: lenguaje de máquina, lenguaje ensamblador, C, Fortran, Pascal, C++, C#, entre otros.

#### Paradigma Paradigma estructurado:

Es un caso particular del PI, porque se impone que las instrucciones sean agrupadas en bloques (procedimientos y funciones); entre otras características. El código tien la forma de un ciclo, gobernado por una condición lógica.

#### Paradigma declarativo:

Las acciones para solucionar y describir el problema son aleatorias. Ejemplo: SQL, JAVA, LISP. Tenemos dos casos particulares:

#### Paradigma Funcional:

Todo se escribe como una función.

#### Paradigma Logico:

Todo se escribe como un predicado lógico.

#### Paradigma orientado a objetos

Algunos ejemplos: C++, JAVA, C#, Python. Tenemos tres tipos principios:

#### Encapsulación:

Este principio se encapsulan datos, estados y operaciones.

#### Prototipos, clase y herencias:

Los dos primeros son las abstracciones del objeto. Pueden haber prototipos que hereden y definan nuevos objetos.

#### Tipificación y polimorfismo:

Constituyen la comprobación del tipo con respecto a la jerarquía de las clases.

#### Paradigma de programación por eventos:

Un programa se concibe como una iteración infinita con dos objetivos: detectar los eventos y establecer el cálculo capza de tratar el evento.

#### Paradigmas paralelo, distribuido y concurrente:

Una característica

- Un programa no se realiza con una sola unidad de cómputo. En este caso, el programa se corta en subprogramas o rutinas que se ejecutan de manera independiente sobre otras unidades de cómputo (sincrónico o asincrónicamente).

## 2. Repaso

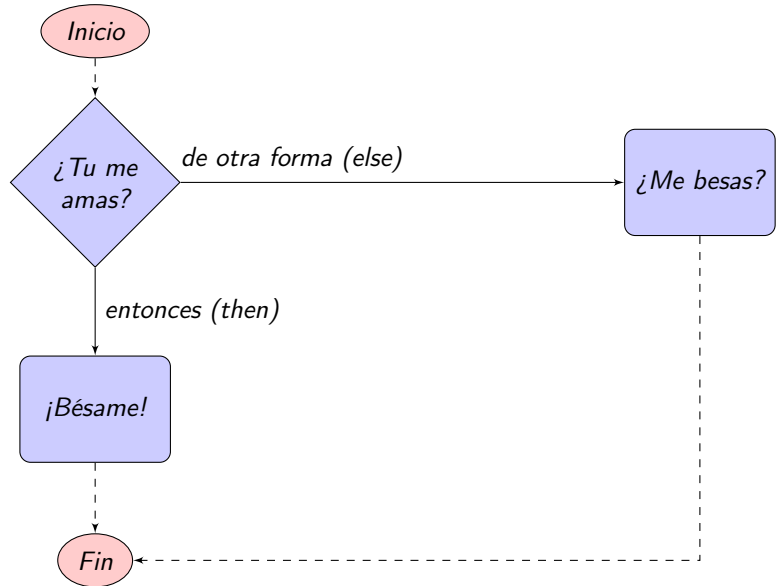
Haremos algunos ejercicios con python y diagramas de flujo

### 2.1. Pseudocódigo

Vamos a usar algunas líneas para mostrar el pseudocódigo implementado en L<sup>A</sup>T<sub>E</sub>X (tomado de <http://minisconlatex.blogspot.mx/2012/10/pseudocodigo.html>)

**Ejemplo 2.1.** Un ejemplo del pseudocódigo y diagrama de flujo para *if*.

```
if Tu me amas then
|   ¡Bésame!
else
|   ¡Me besas?
end
```



### 2.2. Python

Un "Hola mundo" en python:

**Ejemplo 2.2.** Ejemplo de un código en Python

```
1 >>> print "Hola mundo"
```

Python nos puede ayudar como herramienta para realizar programas en C.

### 2.3. Diagramas de flujo

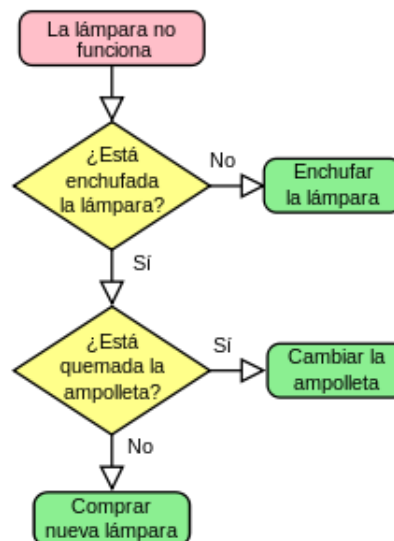
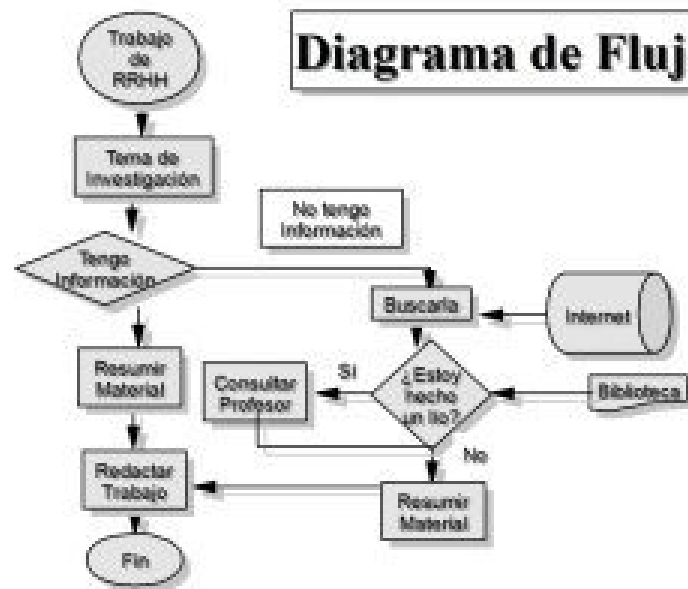


Figura 3.3: Tomado de Wiki



Además tendrás una lectura para revisar en SAE.

## Parte IV: Introducción a la programación en C

### Capítulo 4 La programación

#### Objetivo

Explicar los conceptos de programación de las computadoras así como el proceso de desarrollo de un programa.

#### 1. Lenguajes de programación

En esta sección mostraremos algunos comandos, funciones y generalidades relacionadas a los lenguajes de programación; no obstante, nuestro interés es inclinarnos hacia la programación en C.

Algunos lenguajes de programación: C, C++, Python y Java.

##### 1.1. Definiciones (programación, programa, lenguaje)

En las siguientes líneas se muestran las líneas de un código en C.

##### Ejemplo 1.1. Partes de un programa

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6 // Funcion principal
7
8 int main()
9 {
10     int a, b, c; // Declaracion de variables
11
12 // Instrucciones
13
14
15 // Regresa un dato de acuerdo al tipo de la funcion principal
16     return 0;
17
18 }
```

Existen las variables globales que son ...

A diferencia de las variables locales las cuales desaparecen cuando termina el bloque donde están definidas, la memoria asignada para las variables globales permanece asignada a través de la ejecución del programa.

##### Ejemplo 1.2. Partes de un programa

```

1
2 #include // Directivas de preprocesador. Pueden ser funciones y datos
   predefinidos.
3
4 #define // Macros del procesador
5
6
7 main() // Los parentesis indica que es un bloque de construccion denominado
   funcion
8 {
9
10 // Dentro de esta funcion principal se definen
11 // las declaraciones locales; las variables locales.
```

```

12 }
13 }
14
15
16 // Se pueden definir otras funciones, recuerdas...

```

Algunas definiciones

Un **programa** es un conjunto de instrucciones, que una vez ejecutado, realiza una o varias tareas en una computadora [Juganaru Mathieu \(2012\)](#).

La programación es el acto de escribir programas siguiendo un lenguaje (o varios) de programación y respetando la **sintaxis** (reglas de escritura, forma) y la **semántica** (conjunto de reglas, significado).

## 1.2. Traductores del lenguaje (compiladores e intérpretes)

Un **compilador** es un programa que traduce los programas fuentes escritos en lenguaje de alto nivel (C, Fortran) a lenguaje de máquina [L. y Zahonero Martínez \(2001\)](#).

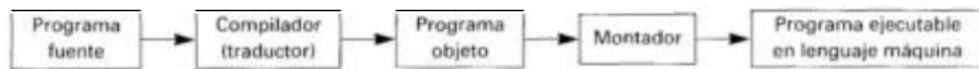


Figura 4.1: Fases de la compilación. Tomado de [L. y Zahonero Martínez \(2001\)](#)

Un **intérprete** es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta. [L. y Zahonero Martínez \(2001\)](#).



Figura 4.2: El intérprete. Tomado de [L. y Zahonero Martínez \(2001\)](#)

## 2. Algunos conceptos de programación

### 2.1. Identificadores

Un **identificador** es una secuencia de letras, dígitos, y subrayados. Las letras mayúsculas y minúsculas son distintas (C es sensible a las mayúsculas) [L. y Zahonero Martínez \(2001\)](#).

Algunos ejemplos de identificadores son:

nombre\_clase, Indice, Fecha\_Compra\_Casa, Habitacion120 entre otros. Es importante notar Algunas palabras están reservadas por el lenguaje. Consejos para para escribir sus identificadores:

1. Identificadores de variables con minúscula
2. Constantes en Mayúsculas
3. Funciones con tipo de letra mixto: mayúscula/minúscula

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

Figura 4.3: Algunos identificadores reservados por el lenguaje C. Tomado de [L. y Zahonero Martínez \(2001\)](#)

## 2.2. Secuencias de escape

Código de escape	Significado	Códigos ASCII	
		Dec	Hex
'\n'	nueva línea	13 10	0D 0A
'\r'	retorno de carro	13	0D
'\t'	tabulación	9	09
'\v'	tabulación vertical	11	0B
'\a'	alerta (pitido sonoro)	7	07
'\b'	retroceso de espacio	8	08
'\f'	avance de página	12	0C
'\\'	barra inclinada inversa	92	5 c
'\''	comilla simple	39	27
'\"'	doble comilla	34	22
'\?'	signo de interrogación	63	3F
'\000'	número octal	<i>Todos</i>	<i>Todos</i>
'\xhh'	número hexadecimal	<i>Todos</i>	<i>Todos</i>

Figura 4.4: Secuencias de escape. Tomado de [L. y Zahonero Martínez \(2001\)](#)

## 2.3. Los formatos

Los códigos de formatos más utilizados:

%d	el dato se convierte a entero decimal
%o	el dato se convierte a octal
%x	el dato se convierte a hexadecimal
%u	el dato se convierte a entero sin signo
%c	el dato se considera de tipo carácter
%e	el dato se considera de tipo float, notación científica
%f	el dato se considera de tipo float, notación decimal
%g	el dato se considera de tipo float: %e o %f, busca la más corta.
%s	el dato ha de ser cadena de caracteres
%d	el dato se considera de tipo double

## 2.4. Operadores

### • Operadores de asignación y unarios

$=$       $a = b$     asigna el valor de  $b$  a  $a$   
 $*$   $=$       $a * b$     Multiplica  $a$  por  $b$  y asigna el resultado a la variable  $a$   
 $/$   $=$       $a / b$     Divide  $a$  entre  $b$  y asigna el resultado a la variable  $a$   
 $\%$   $=$       $a \% b$     asigna a  $a$  al resto de  $a/b$   
 $+$   $=$       $a + b$     suma  $b$  y  $a$  y lo asigna a la variable  $a$   
 $-$   $=$       $a - b$     resta  $b$  de  $a$  y lo asigna a la variable  $a$   
 $\pm \pm a$     $a = a \pm 1$    opera sobre un solo operando para producir un nuevo valor

### • Operadores relacionales

$==$     $a == b$     $a$  igual a  $b$   
 $!=$     $a != b$     $a$  no igual a  $b$   
 $>$      $a > b$      $a$  mayor que  $b$   
 $<$      $a < b$      $a$  menor que  $b$   
 $>=$     $a >= b$     $a$  es mayor o igual que  $b$   
 $<=$     $a <= b$     $a$  es menor o igual que  $b$

### • Operadores lógicos (booleanos)

$!$          $!$     no        Alta prioridad  
 $||$        $a || b$     $a$  o  $b$     Media prioridad  
 $\&\&$      $a \&\& b$     $a$  y  $b$     Baja prioridad

El operador  $!$  es falso para si el operando es verdadero, distinto de cero; el operador  $||$  es verdadero si cualquiera de los operandos es verdadero; el operador  $\&\&$  es verdadero sólo si ambos operandos son verdaderos.

Tenga cuidado con las diferencias que hay entre  $++n$  (prefijo) y  $n++$  (sufijo) (incrementa  $n$  en 1 y lo asigna a  $m$ ; en el segundo caso lo asigna con su valor inicial hace algo, depende del contexto, lo imprime [si es el caso] y luego lo incrementa [L. y Zahonero Martínez \(2001\)](#)).

Ver github o el siguiente archivo

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 /*
6 Tenga cuidado con las diferencias que hay entre ++n y n++
7 (incrementa n en 1 y lo asigna a m; en el segundo
8 caso lo asigna con su valor inicial lo imprime [si es el caso] y luego
9 lo incrementa)
10 */
11
12
13
14 #include <stdio.h>
15 int main()
16 {
17     int a = 11;
18     int b = 10;
19     float c = 10.5;
20     float d = 100.5;
21
22     if(a!=11){
23         printf("Algo anda mal a = %d\n", ++a);
24     }
25     else
26     {
27         printf("Todo bien a = %d\n", ++a);

```

```

28 };
29
30 if(b!=9){
31     printf("Algo anda mal b = %d\n",--b);
32 }
33 else
34 {
35     printf("Todo bien b = %d\n", --b);
36 };
37
38
39 if(c!=10.5){
40     printf("Algo anda mal c = %f\n",++c);
41 }
42 else
43 {
44     printf("Todo bien c = %f\n", ++c);
45 };
46
47
48 if(d!=99.5){
49     printf("Algo anda mal d = %f\n",--d);
50 }
51 else
52 {
53     printf("Todo bien d = %f\n", --d);
54 };
55
56     return 0;
57 }
58
59 // El operador ++ o -- incrementa (reduce)
60 // el valor de la variable en 1

```

### • Operadores especiales

() es el operador de llamada a funciones.

[] dimensiona arreglos y designa un elemento en un arreglo (*array*)

sizeof produce un resultado que es el tamaño de (bytes) del dato especificado.

El operador sizeof es un operador unitario: opera sobre un valor único.

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6 int main(){
7
8
9     printf("Doble: %d\n", sizeof (double));
10    printf("Entero: %d\n", sizeof (int));
11    printf("flotante: %d\n", sizeof (float));
12    printf("Caracter: %d\n", sizeof (char));
13
14    return 0;
15 }

```

### • Operadores de direcciones

\* Lee o modifica el valor apuntado por la expresión

& Permite acceder a un miembro de un dato agregado.

## 2.5. Algunos comentarios generales

La función gets() es de cuidado:



```
galileo gcc -Wall 9gets.c -o 9gets
9gets.c:9:6: warning: return type of 'main' is not 'int' [-Wmain]
void main()
    ^
9gets.c: In function 'main':
9gets.c:14:3: warning: implicit declaration of function 'gets' [-Winimplicit-function-declaration]
  gets(cadena);
  ^
/tmp/ccIfCXH0.o: In function 'main':
9gets.c:(.text+0x2e): warning: the 'gets' function is dangerous and should not be used.
galileo
galileo gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figura 4.5: Mensaje de salida con la función `gets()` en Ubuntu 16.04. La otra imagen muestra la versión del compilador.

El operador “\*” tiene prioridad alta frente a la suma. Usa los paréntesis. Operadores con la misma prioridad usan asociatividad de izquierda a derecha [L. y Zahonero Martínez \(2001\)](#).

Las cadenas de caracteres no pueden compararse directamente, debido que la comparación se realiza entre las direcciones de memoria de las cadenas. [L. y Zahonero Martínez \(2001\)](#).

En la sección 5 se exponen los conceptos relacionados a las funciones de biblioteca.

### 3. Pasos para crear un programa

Algunos autores presentan una lista de pasos para crear (solucionar) un problemas usando la computadora.

1. Definir el programa: análisis.
2. Definir directivas del preprocesador.
3. Definición de decalaracione globales.
4. Crear la función principal.
5. Crear el cuerpo del programa.
6. Crear las funciones definidas por el usuario.
7. Compilar, enlazar, ejecutar y comprobar el programa: corregir errores.
8. Implementar documentación.

## Capítulo 5

### Tipos de datos

Mostraremos algunos tipos datos básicos. Los requisitos de memoria para cada tipo de datos puede variar de un compilador a otro. Para esta parte usaremos varios libros y aparecerán a lo largo del documento, en la parte final encuentras todas las referencias. Puedes revisar más información en [Gottfried \(1996\)](#); [Kochan \(2004\)](#).

#### 1. Representación básicos

Veamos en lenguaje C, los tupos de datos:

##### Código en C. 1.1. Tipos de datos:

```

1 /*Tipo de dato| Descripcion      | Requisito de memoria */
2
3 int /* Cantidad entera          | 2 bytes o una palabra */
4
5 char /* Caracter                | 1 byte    */
6
7 float /* Numero que incluye     | 4 bytes   */
8 /* punto decimal o exponente    */
9
10 double /* Numero en coma flotante 8 bytes */
11 /* de doble precision           */

```

Un dato tipo char se utiliza para datos individuales. Este requiere solamente un byte de memoria, o sea,  $2^8$ . Es decir un dato de este tipo podrá tomar valores de 0 a 255. Aunque algunos compiladores representan los datos de este tipo de -128 a +127.

Se pueden usar cualificadores de tipos de datos:

### Código en C. 1.2. Cualificadores:

```

1 /* Tipo de dato con cualificador */
2
3 short int /* requiere menos memoria o la misma que uno int*/
4
5 long int  /* requiere la misma o mayor memoria, pero no menos que uno int*/
6
7 unsigned int /* requiere la misma memoria que un int. Todos los bits se reservan
8 para el valor numerico. No requiere que el bit mas a la
9 izquierda se reserve para el signo*/
10
11 unsigned short int /*Puede existir este tipo de dato*/

```

donde se muestran los cualificadores para enteros, no obstante la interpretación de los datos con calificador puede variar de un compilador a otro.

Un dato entero sin signo (`unsigned int`) tiene 2 bytes esto es: 16 bits; esto significa que tenemos  $2^{16}$  podría tomar valores de 0 a 65535, mientras que un entero con signo podría variar de: -32768 a +32767. El cualificador `unsigned` puede usarse para el tipo `char`.

La aplicación de otros tipos de cualificadores dependen de los compiladores.

Estos son los tipos básicos de memoria (en lenguaje C), más adelante veremos otros tipos de datos.

Lo anterior es importante porque cada identificador que representa un número o caracter dentro de un programa debe tener asociado un tipo básico antes de que el identificador aparezca en una instrucción ejecutable.

Las declaraciones de los objetos en C requiere el tipo de otra forma el compilador muestra errores. Por otro lado, cada objeto tiene las siguientes propiedades:

## 2. Propiedades de cada objeto en C

- Alcance (ámbito):  
Es la región del código donde una declaración de un objeto está activa o existe.
- Visibilidad:  
Es la región del código donde un objeto está activo y no oculta, o es escondido, otro objeto que tiene el mismo identificador.
- Durabilidad:  
Este concepto se asocia con el tiempo de ejecución de un programa donde el objeto existe en la memoria.
  - Durabilidad Estática: Es aquella en la que el objeto perdura desde la compilación hasta el final. Ejemplo: funciones declaradas y variables declaradas con `static`.

- **Durabilidad Local:** Es aquella en la que el objeto es creado en la entrada de un bloque y se borra a la salida. Ejemplo: variables declaradas con `auto`
- **extern** Estos objetos tienen durabilidad `static` e informa al enlazador de programas para que realiza las unificaciones pertinentes entre ficheros.
- **register** Se puede utilizar para variables locales y argumentos de funciones. Hace que el compilador le asocie un acceso de memoria rápido.

Finalmente, mostramos que la forma general de declarar una variable es:

```
durabilidad tipo identificador;
```

### 3. Representación de enteros

En C, tenemos cuatro (4) tipos básicos de constantes.

#### Código en C. 3.1. Tipos de constantes: int

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2
3 entera en los diferentes Sistemas de Numeracion
4
5 Este tipo de dato: Representa numeros. Tipo numerico
6
7 */
8
9
10 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
11 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
12 /*Mostraremos ejemplos errados de...*/
13
14 /* Constante entera decimal. No empieza con 0 */
15
16 d1=12,245;
17 d5=0900;
18 d2=36.0;
19 d3=10 20 30;
20 d4=12-45-6789;
21
22
23 /* Constante entera octal. Empieza con 0 */
24
25 o1=743;
26 o2=05280;
27 o3=0777.777;
28
29 /* Constante entera hexadecimal. Empieza por 0x o 0X*/
30
31 h1=0X12.34;
32 h2=0BE38;
33 h3=0x.4bff;
34 h4=0XDEFG;
35 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
36 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
37 /*Mostraremos ejemplos errados de...*/
```

#### Código en C. 3.2. Tipos de constantes: float

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 en punto (coma) flotante
3
4 */
```

```

5 /* Representan numeros. Tipo numerico */
6
7 /* Ejemplos de en punto (coma) flotante */
8
9 f1=0.;
10 f2=1.;
11 f3=0.2;
12 f4=321.1307;
13 f5=1E6;
14 f6=1E+6;
15 f7=0.01E-3;
16 f8=0.2e-7;
17
18 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
19 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
20 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
21 /*Mostraremos ejemplos errados de en coma flotante*/
22 ff1=0,1;
23 ff2=1;
24 ff2=1.
25 ff3=1E+10.1;
26 ff4=3e 10;
27 /*Mostraremos ejemplos errados de en coma flotante*/
28 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
29 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
30 /* Atencion-Atencion-Atencion-Atencion-Atencion-Atencion-*/
31
32 /*Las siguientes cantidades corresponden a misma y estan escritas de forma
    correcta*/
33
34
35 FF1=300000.;
36 FF2=0.3E6;
37 FF3=0.3e6;
38 FF4=3E5;
39 FF5=3E+5;
40 FF6=3.0E+5;
41 FF7=3.0e+5;
42 FF8=3.0E+5;
43 FF9=30.E+4;
44 FF10=300e3;

```

### Código en C. 3.3. Tipos de constantes: char

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2
3 de caracter /* Este tipo de constante se encierra en comillas simples*/
4
5 */
6
7 /*Los siguientes son ejemplos de tipos char*/
8
9 c1= 'h';
10 c2= 'o';
11 c3= 'l';
12 c4= 'a';
13 c5= ' ';
14 c6= 'm';
15 c7= 'u';
16 c8= 'n';
17 c9= 'd';
18 c0= 'o';

```

El valor máximo de una constante entera varía de una computadora a otra.

Suponiendo que una computadora utiliza una palabra de  $n$  bits. Entonces, una cantidad entera (`int`)  $x$ , estará en el rango:  $-2^{n-1} \lesssim x \lesssim +2^{n-1} - 1$ . y una cantidad sin signo (`unsigned int`)  $y$ , estará en el rango:  $0 \lesssim y \lesssim 2^n - 1$ . Cambiaremos  $n \rightarrow \frac{n}{2}$  para enteros cortos y  $n \rightarrow 2n$  para enteros largos (`long int`). Pero recordemos que las reglas pueden cambiar de acuerdo a cada computadora.

Las constantes de tipo entera y en coma flotante tienen las siguientes reglas:

1. No se pueden incluir comas ni espacios en blanco en la constante.
2. Si se desea, la constante puede ir precedida de un signo menos (-). (Realmente, el signo menos es un operador que cambia el signo de una constante positiva, aunque se puede ver como parte de la constante misma.)
3. El valor de una constante no puede exceder un límite máximo y un mínimo especificados. Para cada tipo de constante, estos límites varían de un compilador a otro.

Las constantes largas sin signo no pueden ser negativas y se identifican con la letra U al final, por ejemplo: 50000U (sistema decimal, sin signo), 0123456L (sistema octal, larga) y 0xFFFFFUL (hexadecimal, sin signo y larga)

#### 4. Constantes simbólicas

Este tipo de constantes se definen al inicio del programa.

Este tipos de constantes constituyen una cadena de caracteres, que pueden representar una constante numérica, una constante de caracter, o una cadena de caracteres. Este tipo de constante permite que aparezca un NOMBRE en lugar de una constante numérica, una constante de caracter o una cadena de caracteres.

Veamos un ejemplo:

##### Código en C. 4.1. Constantes simbólicas

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6 int x,y,z;
7
8 int main(){
9
10 printf("Dame el primer numero: ");
11
12 scanf("%d",&x);
13
14 printf("\n");
15
16 printf("Dame el segundo numero: ");
17
18 scanf("%d",&y);
19
20 z=x+y;
21 printf("\n\n El resultado de la suma es :%d\n",z);
22 return 0;
23 }
```

Regresaremos a este tema posteriormente.

## Capítulo 6

### El lenguaje de programación

#### Objetivo

Implementar algoritmos en lenguaje C para la solución de problemas computables.

#### 1. Conversiones

Existen dos tipos de conversiones:

##### 1. Conversión Implícita:

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>
6
7 /*
8 Algunos notas sobre los tipos de constantes
9 */
10
11
12 int main(){
13
14 int a=11;
15 float b=4.0;
16
17
18 b = b + a;
19 printf("El valor de a se convierte en float (%f) antes de la suma\n", b);
20
21
22 b= a/5.;
23 printf("El valor de a se divide y queda float (%f) despues se convierte a
    float y se asigna a b\n", b);
24
25
26 b= b/5;
27 printf("Toma el valor anterior de b, Convierte el valor 5 a tipo float, hace
    una division (%f) y despues lo asigna a b\n", b);
28
29 return 0;
30 }

```

##### 2. Conversión Explícita:

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>
6
7 /*
8 Este programa muestra la forma de realizar conversiones explicitas de
9 tipo de constantes.
10 */
11

```

```

12
13 int main(){
14
15 int a = 11;
16 double b = 4.6;
17
18
19
20 printf("\n Usando el formato (tiponombre)valor, podemos cambiar el tipo de
    dato (CAST).\n");
21
22 printf("Por ejemplo a = %i, para cambiar a float, usamos el formato (float)a
    = %f \n\n", a, (float)a);
23
24
25 printf("Por ejemplo b = %f, para cambiar a entero, usamos (int)b =
    %i \n\n", b, (int)b);
26
27
28 return 0;
29 }

```

## 2. Sentencias y operadores

El condicional (**if**) más simple:

### Forma general de una declaración if

```
if(condicion){instrucciones}
```

La sentencia **if-else**

### Forma general de una declaración if

```

if(condicion1)
    {instrucciones si condicion1 es verdadera;}else
    {instrucciones si condicion1 es falsa;}

```

Condional anidado:

**Ejemplo de una declaración if-else-if**

```

if (condicionA)
{
    Instrucciones si la condicionA es verdadera
                        la condicionB es falsa y...
    if (condicionA1)
    {
        ...si la condicionA1 es verdadera;
    }
    else
    {
        ...si la condicionA1 es falsa;
    }
}
else if (condicionB)
{
    Instrucciones si la condicionA es falsa
                        la condicionB es verdadera y...
    if (condicionB1)
    {
        ...si la condicionB1 es verdadera;
    }
    else
    {
        ...si la condicionB1 es falsa;
    }
}
else if (condicionC)
{
    Instrucciones si la condicionA y condicionB son falsas
                        y la condicionC es verdadera y...
    if (condicionC1)
    {
        ...si la condicionC1 es verdadera;
    }
    else
    {
        ...si la condicionC1 es falsa;
    }
}
}

```

En esta sección encontraremos diferentes tipos de sentencias. veremos algunos ejemplos:  
 Veamos un ejemplo para la sentencia while:

**Forma general de una declaración while**

```
while(condicion){instrucciones}
```

**Ejemplo 2.1.** *Un ejemplo usando una sentencia do-while:*

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5
6 #include <stdio.h>
7

```



```

8 int main(){
9
10 int DigIni, DigFinal;
11
12 printf("Dame un valor entero inicial ");
13
14 scanf("%i",&DigIni);
15 DigIni = DigIni - 1;
16
17
18 printf("Dame un valor entero final ");
19
20 scanf("%i",&DigFinal);
21 DigFinal= DigFinal - 1;
22
23 printf("Ahora te muestro una lista de datos:\n");
24
25
26 while(DigIni <= DigFinal)
27 {
28 DigIni= DigIni+1;
29 printf(" %d\n",DigIni);
30 }
31 return 0;
32 }

```

#### Forma general de una declaración while

```
do{instrucciones}while(condicion)
```

**Ejemplo 2.2.** *Un ejemplo usando una sentencia do-while:*

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6
7 int main(){
8
9 int digito=0, DigFinal=5;
10
11 printf("Ahora te muestro una lista de datos:\n");
12
13 do
14 {
15 printf(" %d\n", ++digito);
16 }
17 while(digito <= DigFinal);
18 return 0;
19
20 }

```

#### Forma general usando un operador ternario

```
A ? B : C
```

### Forma general de una declaración for

```
for(declaracion_inicializacion; expresion_prueba; expresion_siguiete)
{
    instrucciones (sentencias);
    ...;
    ...;
}
```

La **declaracion de inicialización** se ejecuta una vez se evalúa la **expresion de prueba** y si es falsa, entonces se termina el bucle, de otra forma las sentencias se ejecutan hasta que la **expresion siguiente** se cumple. El proceso se repite hasta que la **expresion de prueba** es falsa.

#### Ejemplo 2.3. Instrucción for.

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 /*
5 declaracion for
6 */
7
8
9 #include <stdio.h>
10
11
12 int main(){
13 int i;
14
15 for(i=0; i<=5; i++)
16 {
17 printf("los digitos son: %i\n", i++);
18 }
19
20 return 0;
21 }
```

Este tipo de sentencia (for) se implementa cuando conocemos el numero de veces que deseamos repetir la accion (iteraciones) y es una de las mas usadas en diferentes lenguajes de programacion.

### 3. Control de bucles

En esta parte mostramos algunos ejemplos de los bucles y funciones que nos sirven para evitar errores. En general las banderas y centinelas son conocidos como interruptores; y a veces conmutadores.

### 4. Funciones de usuario

Estas son las funciones que también llamamos, **funciones definidas (implementadas o diseñadas por el usuario)**

A estas funciones le relacionamos los siguientes conceptos importantes:

- Definición: Da nombre y reserva un espacio de memoria. Para llamar una función usamos:
- Declarada: También llamada como prototipo. Se colocar abajo de los archivos de cabecera.
- Llamada: Invoca la función. La forma general de llamar una función es:

```
Nombre_de_la_Funcion(Lista_de_Parametros)
```

En esta parte los parámetros deben ser numéricos.

En general...

Una **función** es un subprograma que devuelve, un valor un conjunto de valores o realiza alguna tarea específica como I/O L. y Zahonero Martínez (2001).

También hay funciones que pueden ser definidas por el usuario. Las funciones son conceptos muy importantes en el paradigma de programación en un “lenguaje de programación estructurado” como C.

Las funciones pueden ser **llamadora** y función **llamada** L. y Zahonero Martínez (2001).

Además una función debe ser definida y puede ser declarada. En las siguientes líneas trabajaremos sobre estos y otros conceptos relacionados a las funciones.

#### Ejemplo 4.1. Forma general de una función en C.

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5
6 float suma(float x, float y){
7
8 return x+y;
9
10 }
```

No es aconsejable llamar una función dentro de otra. Todo código debe ser listado secuencialmente L. y Zahonero Martínez (2001).

La declaración de una función se denomina **prototipo**.

Una **declaración** es proporcionar un nombre y darle características y una **definición** es proporcionar un nombre a una entidad y reservar un espacio de memoria para esa entidad L. y Zahonero Martínez (2001).

Una **declaración** de una función contiene solamente la cabecera de la función y una vez declarada la función, la **definición** completa de la función debe existir en algún lugar del programa, antes o después de main()

Un **procedimiento** es una función que no devuelve resultado y su retorno es un void L. y Zahonero Martínez (2001).

El intercambio de parámetros se realiza por medio del concepto paso por valor (o paso por copia), que consiste en enviar una copia a la función llamada. Veamos un ejemplo con código:

#### Ejemplo 4.2. Paso por copia.

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6
```

```

7 void func(int j){
8
9 printf("%i\n", j);
10
11 }
12
13
14 int main(){
15
16 int j=6;
17
18 func(j);
19 return 0;
20 }

```

Diagrama del paso por copia:

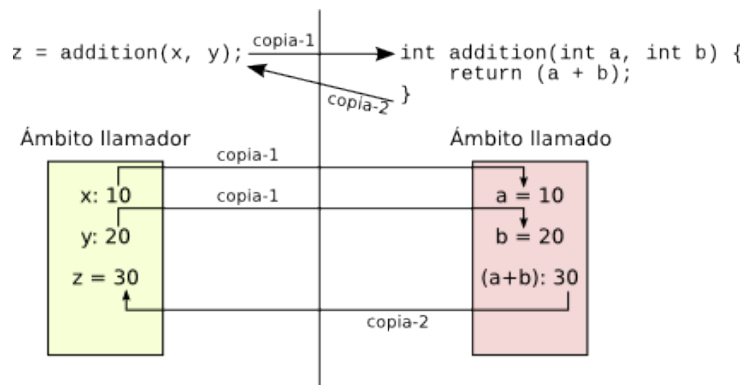


Figura 6.1: Imagen que muestra el ejemplo del paso por copia. Tomada de: UC3M

El **paso por valor o copia** significa que la función llamada recibe una copia de los valores de los parámetros, luego procesa la información y envía la información de regreso a la función llamadora (donde se invoca la función llamada) (L. y Zahonero Martínez, 2001).

#### 4.1. Apuntadores

Un apuntador es una variable cuyo valor es la dirección de alguna otra variable, i. e., la dirección directa de la ubicación de la memoria. Como cualquier variable o constante, se debe declarar un puntero antes de usarlo para almacenar la dirección variable. La forma general de la declaración variable apuntador es: `tipo *nombre-de-variable`. Donde `tipo` es uno de los tipos básicos y `*nombre-de-variable` será el nombre dado. L. y Zahonero Martínez (2001).

#### 4.2. Pasos de parámetros por referencia

Cuando se desea que una función modifique el valor del parámetro y devolver este valor modificado a la función llamadora, se utiliza el método de paso de parámetro por dirección. En este método el compilador pasa la dirección de memoria del valor del parámetro a la función. Para usar este método se debe usar el símbolo `&` antes del nombre de la variable y el parámetro variable correspondiente de la función debe declararse como puntero.

**Ejemplo 4.3.** Paso por referencia.

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3

```

```

4 #include <stdio.h>
5 void intercambio(int *a, int *b){
6     int aux = *a;
7     *a=*b;
8     *b=aux;
9 }
10
11 int main(){
12     int i=3, j=50;
13     printf("i=%d y j=%d\n", i,j);
14     intercambio(&i, &j);
15     printf("i=%d y j=%d\n", i,j);
16     return 0;
17 }

```

Veamos lo que está realizando el algoritmo anterior. En la primer columna tenemos dirección de memoria y en la segunda un identificador.

		aux=*a		*a=*b		*b=aux	
0X001	a	0X003	a	0X002	a	0X002	a
0X002	b	0X002	b	0X003	b	0X001	b
0X003	aux	0X001	aux	0X001	aux	0X003	aux

Otra forma de ver el algoritmo anterior.

		aux=*a		*a=*b		*b=aux	
0X001	a	0X001	aux	0X001	aux	0X001	b
0X002	b	0X002	b	0X002	a	0X002	a
0X003	aux	0X003	a	0X003	b	0X003	aux

### 4.3. Algunos usos de los apuntadores

#### Ejemplo 4.4. Algunos usos de los apuntadores

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>
6
7 int main(){
8
9     int var = 20;
10    int *ip;
11    ip = &var;
12
13    printf("valor var = %i (entero)\n", var);
14    printf("valor var = %x (hexadecimal)\n", var);
15    printf("valor *ip = %i (entero)\n", *ip);
16    printf("valor *ip = %x (hexadecimal)\n", *ip);
17    return 0;
18 }

```

Veamos un esquema:

0X001	var	20	0X001	var	20
0X002	ip	&var	0X002	ip	0X001

Con el ejemplo anterior vemos la forma de obtener la dirección de memoria y el o los caracteres guardados en una dirección de memoria a través del concepto de indirección.

#### 4.4. Apuntadores nulos

Esta es una forma de inicializar un apuntador.

**Ejemplo 4.5.** *Un código para un apuntador nulo.*

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>
6
7 int main(){
8
9 int *ip;
10
11 ip = NULL;
12
13 printf("valor var = %p\n", ip);
14
15 return 0;
16 }

```

#### 4.5. Funciones en línea, macros con argumentos

El uso de funciones implementadas por el usuario, aumenta la velocidad del programa. En esta parte estudiaremos las funciones en línea. Aunque su nombre no es adecuado ya que el preprocesador expande o sustituye la expresión cada vez que es llamada. El compilador inserta el código en el punto en que esta función en línea o macros con argumentos se llama.

En estas líneas definiremos la función creada por el usuario en las directivas del preprocesador:

**La forma general es**

```
#define NOMBRE_MACRO(parametros_NO_tipo) expresion-texto
```

El problema con este tipo de macros es que se consume mucha memoria ya que la toda invocación del macro representa una copia de la expresión completa que representa.

Veamos un ejemplo de un macro con argumentos de varias líneas.

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5 #define fesp(x) (x*x +2*x - 1)
6
7 int main(){
8
9 float x;
10
11 for (x =0.0; x<= 6.5; x +=0.5)
12 {
13 printf("f(%.1f) = %6.2f\n", x, fesp(x));
14 }
15
16 return 0;
17 }

```

### 5. Funciones de biblioteca

Las funciones de biblioteca son parte de las bibliotecas colocadas en los archivos de cabecera. Las funciones realizan diferentes operaciones, incluso algunas dependientes de la computadora. Por ejemplo, las funciones de entrada y salida, abrir y cerrar archivos o aquellas que realizan cálculos matemáticos [Gottfried \(1996\)](#).

Ya hemos visto varias funciones de biblioteca (y otras se retom), ahora incluiremos otras, por ejemplo: `sin()`, `cos()` entre otras.

**Ejercicio 5.1.** Realiza una investigación sobre diez (10) bibliotecas y al menos dos (2) funciones de bibliotecas asociadas. Sube los códigos para subirlos en el sitio sea en el foro de la pestaña **Funciones y funciones de biblioteca**. Tendremos una sesión para exponer los código e implementarlos.

Las **funciones de biblioteca** o **funciones predefinidas** son aquellas pertenecientes a la biblioteca estándar, se dividen en grupos y todas las que pertenecen al mismo grupo se declaran en el mismo archivo de cabecera L. y Zahonero Martínez (2001).

Algunos nombres de los archivos de cabecera estándar usados en los programas que realizamos son:

$$\begin{array}{ll} < \text{assert.h} > & < \text{ctype.h} > \\ < \text{time.h} > & < \text{signal.h} > \end{array} \quad (6.1)$$

Más bibliotecas pueden encontrarse en [tutorialspoint.com](http://tutorialspoint.com)

<code>tolower()</code>	Convierte a letras minúsculas
<code>isdigit()</code>	comprueba si c es un dígito de 0-9, verdadero en este caso o falso en caso contrario
<code>toupper()</code>	convierte a letras mayúsculas.
<code>isalnum()</code>	Devuelve verdadero, si c es un dígito 0-9 o un caracter alfabético (M o m) y falso en cualquier otro caso.

Podemos crear programas para implementar números (pseudo) aleatorios. Usaremos una semilla como un entero que nos ayuda a obtener diferentes valores de nuestros números aleatorios.

Instrucción que inicializa el generador de números aleatorios

```
srand(time(NULL));
```

#### • Funciones numéricas

La mayoría de las funciones numéricas están en el archivo de cabecera `<math.h>`.

Algunas funciones matemáticas son:

<code>ceil()</code>	Redondea al entero más cercano.
<code>fabs()</code>	Devuelve el valor absoluto de x
<code>floor()</code>	Redondea al entero más próximo.
<code>sqrt()</code>	Devuelve la raíz cuadrada

Además de otras funciones como: `asin()`, `acos()`, `log()`, `log10()`, entre otras.

Algunas veces necesitamos generar números aleatorios, por lo que existen las funciones:

<code>rand()</code>	Genera un número (pseudo) aleatorio. Requiere <code>&lt;stdlib.h&gt;</code>
<code>randomize()</code>	inicializa el generador de números aleatorios con una semilla aleatoria obtenida a partir de una llamada a la función <code>time</code> . Requiere <code>&lt;time.h&gt;</code> .

Vamos a retomar algunos conceptos compartidos en la sección 2.

## 6. Tipos de almacenamiento

En particular revisaremos el concepto relacionado con el almacenamiento: la durabilidad (ver subsección 2): “Este concepto se asocia con el tiempo de ejecución de un programa donde el objeto existe en la memoria.” A continuación hablaremos de cuatro tipos de almacenamiento de las variables en C.

- **Tipo de almacenamiento: auto**

Este tipo de variables no existen al inicio de la ejecución de un programa: se crea en algún punto durante la ejecución. Desaparece antes de que el programa termine de ejecutarse.

Algunas características:

1. Se guarda en la memoria
2. El alcance se limita al bloque donde está definido.
3. En caso de no estar definido, el valor inicial que lo contiene es cualquier dato (*garbage*).
4. Permanece activa (la variable) hasta que el control está en el bloque donde la variable se declara.

En general,

Todas las variables locales tienen el tipo de almacenamiento auto.

- **Tipo de almacenamiento: static**

Este tipo de variables se eliminan cuando el programa desaparece de memoria en tiempo ejecución. Ej.: variables globales: por defecto estáticas, visibles globalmente, incluso desde otros archivos (enlace externo).

Algunas características:

1. Se guarda en la memoria
2. El alcance se limita al bloque donde está definido.
3. En caso de no estar definido, el valor inicial que lo contiene es cero.
4. Permanece activa entre las diferentes funciones llamadas.

Veamos un ejemplo:

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3 #include <stdio.h>
4 // Declaracion de la funcion
5 void func(void);
6 static int count = 5; // variable global
7 int main(){
8     while(count--){
9         func(); // llamar la funcion
10    }
11    return 0;
12}
13 // Definicion de la funcion
14 void func( void ) {
15     register int i = 5; //variable local register(static)
16     i++;
17     printf("i es %d y la cuenta es %d\n", i, count);
18 }

```

La clase de almacenamiento static instruye al compilador a mantener la variable local para evitar que la cree y la elimine cada vez que entra o sale del ámbito. Aplicarlo a una variable global hace que su ámbito sea el archivo en el que se declara.

- **Tipo de almacenamiento: extern**

La palabra reservada del lenguaje extern se puede utilizar para notificar al compilador que la declaración del resto de la línea no está definida en el archivo fuente actual. Esta palabra se puede usar para acceder a símbolos externos definidos en otros módulos. Las funciones por defecto son externas por lo que no es necesario usar esta palabra a diferencia de las variables.

Algunas características:

1. Se guarda en la memoria



2. El alcance es global.
3. En caso de no estar definido, el valor inicial que lo contiene es cero.
4. Permanece activo hasta el fin del programa.

Se prefieren variables locales que globales. Podemos usar `static` para hacerla local en relación al archivo en que se define.

Existen las variables de tipo `extern` (no se recomienda usar en funciones definidas por el usuario, porque todas las funciones de este tipo son externas), que se usa para dar referencia de una variable global que es visible en todos los archivos del programa.

Este clase (`extern`) de almacenamiento se usa cuando se tienen múltiples archivos. Ejemplo: (correr con `gcc main.c support.c -o nombredesalida`)

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3 // gcc 04main.c 04support.c
4
5 #include <stdio.h>
6
7 extern void write_extern();
8
9 int count;
10
11 int main(){
12     count = 5;
13     write_extern();
14
15 return 0;
16 }
```

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6 extern int count;
7
8 void write_extern(void) {
9
10     printf("La cuenta es %d\n", count);
11
12 }
```

#### • Tipo de almacenamiento: `register`

Existen las variables de tipo `register` que tienen ventajas de rapidez de manipulación. Este tipo de variables se aconsejan para variables locales. En caso de no tener registros disponibles, entonces se crea la variable local normal.

La variables de tipo `register` se almacenan en un registro del procesador. Estas variables se restringen debido al tamaño y al número.

Una aplicación usual como variables de registro son las variables de control de los bucles o en los condicionales de una sentencia, los cuales se ejecutan a alta velocidad. Ej.: `register int k;`

Esta clase de almacenamiento `register` se usa para definir variables locales que no se quieren almacenar en la RAM. El tamaño máximo es el tamaño del registro. Esto significa que no tiene asignada una ubicación de memoria.

Algunas características:

1. Se guarda en los registros de la CPU.
2. El alcance se limita al bloque donde está contenido.
3. En caso de no estar definido, el valor inicial que lo contiene es cualquier dato (*garbage*).
4. Permanece activo hasta que el control está en el bloque donde la variable se declara.

- **Ejemplos de almacenamiento**

Dejaremos algunos ejemplos:

```

1
2
3 // un ejemplo de una clase de almacenamiento
4 {
5     int a;
6     auto int a;
7 }
8
9 // Debe usarse dentro de funciones.
10
11
12
13
14
15
16
17 // un ejemplo de una clase de almacenamiento
18 {
19     static int a;
20 }
21
22 // Mantiene los valores entre dos funciones llamadas.
23
24
25
26
27 // un ejemplo de una clase de almacenamiento
28 {
29     extern int a;
30 }
31
32 // Se usa para declarar una variable global o funcion en otro archivo.
33
34
35
36
37
38 // un ejemplo de una clase de almacenamiento
39 {
40     register int a;
41 }
42
43 // Se recomienda para aquellas variables de acceso rapido.
```

## 7. Arreglos uni y multidimensionales

Concepto de arreglo se puede extender a otros tipos de datos como struct<sup>1</sup>

Un arreglo (*arrays*, lista o tabla) es una secuencia de datos del mismo tipo. Consta de **elementos** y pueden ser, incluso, estructuras definidas por el usuario.

<sup>1</sup> Los datos de tipo **struct** se retomarán en un curso posterior.

Cada elemento tiene un índice o subíndice que numera consecutivamente de izquierda a derecha, iniciando desde cero (0). Estos índices sirven permitir acceso directo al elemento del arreglo.

Veamos un ejemplo de un arreglo en C:

### Ejemplo 7.1. Ejemplo de un arreglo en C

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5 #define n 100
6
7 void main ()
8 {
9     char nombre[n];
10    char apellido[n];
11
12    printf("Como te gusta que te llamen? ");
13    scanf("%[^\n]%*c", nombre);
14    printf("Cual es tu apellido (sin espacios)? ");
15    scanf("%[^\n]", apellido);
16
17    printf("\v\tHola %s tu apellido es %s\n", nombre, apellido);
18 }
```

### 7.1. ¿Cómo se escriben arreglos en C?

En el siguiente ejemplo mostramos la forma de escribir un arreglo en un archivo fuente y cómo se imprime en terminal tipo vector (v).

### Ejemplo 7.2. Un arreglo de un bloque de cuatro edificios con tres departamentos.

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 int main() {
8     int BloquesDeptos[4][3];
9
10    BloquesDeptos[0][0] = 2;
11    BloquesDeptos[0][1] = 3;
12    BloquesDeptos[0][2] = 1;
13
14    BloquesDeptos[1][0] = 4;
15    BloquesDeptos[1][1] = 5;
16    BloquesDeptos[1][2] = 10;
17
18    BloquesDeptos[2][0] = 12;
19    BloquesDeptos[2][1] = 13;
20    BloquesDeptos[2][2] = 14;
21
22    BloquesDeptos[3][0] = 7;
23    BloquesDeptos[3][1] = 6;
24    BloquesDeptos[3][2] = 8;
25
26    printf("El numero de personas que viven en el departamento 3 del bloque 2 es
27           %i\n", BloquesDeptos[1][2]);
28
29    return 0;
30 }
```

Un arreglo se numera consecutivamente de izquierda a derecha iniciando en "0". Así `a[0]` es el nombre del elemento que está en la primera posición de la izquierda (indexación basada en cero).

Algunas características de los arreglos:

- Un arreglo se declara como sigue: `TIPO Nombre[Numero_elementos]={ele_1, ele_2,...,ele_n}`
- En general el  $i$ -ésimo elemento estará en la posición  $i - 1$ .
- Normalmente, un arreglo se usa para almacenar variables de tipo `char`, `int`, `float`.
- Un arreglo se almacena con sus elementos en una secuencia de posiciones de memoria contigua.

**Ejemplo 7.3.** *Un ejemplo de adquisición de datos por terminal.*

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>      // no requiere biblioteca especial
6
7 int main(){
8
9
10 float a[2]={1, 2};      // TIPO Nom[Long]={ele_1, ele_2,..ele_n}
11
12
13 printf("Tenemos que v[%3.1f %3.1f]\n", a[0], a[1]);
14
15 return 0;
16 }
```

Un ejemplo de adquisición de datos por terminal.

**Ejemplo 7.4.** *Note que en este caso no se requiere el símbolo & (Linux).*

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 #include <stdio.h>
5
6 int main ()
7 {
8
9 char a[10];
10 char b[10];
11
12 printf("Como te gusta que te llamen (sin espacios)? ");
13 scanf("%9s", a);
14
15
16
17 printf("Cual es tu apellido (sin espacios)? ");
18 scanf("%9s", b);
19
20 printf("Hola %s tu apellido es %s\n", a, b);
21
22 }
```

Una cadena (de texto) es un conjunto de caracteres, e.g. BCDEF y un arreglo de caracteres es una secuencia de caracteres: `char cadena[]="BCDEF"`;

Nota que existe las cadenas de caracteres y arreglos de caracteres. La primeras contienen un caractere nulo al final del arreglo de caracteres. Además las cadenas se deben almacenar en arreglos de caracteres, pero no todos los arreglos de caracteres contienen cadenas.

Una cadena en C es un arreglo de caracteres de una dimensión (vector de caracteres) que termina con el carácter especial `'\0'` (cero).

## 7.2. Arreglos bidimensionales

Previamente, revisamos el código 7.2 donde se mostrón un ejemplo de un arreglo bidimensional. Estos son arreglos de arreglos, cuya forma general es:

TIPO\_de\_DATO NOMBRE\_ARREGLO[Num\_filas][Num\_columnas]

**Ejemplo 7.5.** *Un ejemplo de adquisición de datos por terminal.*

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>
6
7
8 int main () {
9
10 static int num = 4;
11
12 char names[4][12] = {"Simpson, B.", "Simpson, H.", "Simpson, M.", "Simpson, E."};
13
14 register int i, j;
15
16
17 for ( i = 0; i < num; i++) {
18 printf("\tLos integrantes de la familia son [%d] = %s\n", i, names[i]);
19 printf("\t\tLa direccion de la memoria de la posicion [%d] es %x\n", i,
20        *(names[i]));
21
22 for ( j = 0; j < 12; j++) {
23 printf("El caracter [%i, %i] es %c \ty su direccion de memoria es %x\n ", i, j,
24        names[i][j], names[i][j]);
25 }
26
27 getchar();
28
29 return 0;
30 }
```

Los indices estan dados de la siguiente forma:

**Ejemplo 7.6.** *Una matriz de  $3 \times 2$ .*

```
int matrix[3][2]={
{{1,1}, {1,2}},
{{2,1}, {2,2}},
{{3,1}, {3,2}},
}
```

Es importante usar los corchetes para los arreglos.

## 8. Concatenación

Concatenación es proceso de agregar una cadena a otra cadena.

ejemplo de código para concatenar cadenas en C:

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 #include <stdio.h>
6 #include <string.h>
7
8 int main () {
9     char fuente1[25], fuente2[25], destino[75];
10
11     strcpy(fuente1, " Esta es la primera fuente. ");
12     strcpy(fuente2, " Este es la segunda fuente. ");
13     strcpy(destino, " Este es el destino. ");
14
15
16     strcat(destino, fuente1);
17     strcat(destino, fuente2);
18
19     printf("Cadena de destino final: %s \n", destino);
20     return 0;
21 }

```

ejemplo de código para concatenar cadenas en python:

```

1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4 a = 'hola '
5 b = 'mundo'
6 print(a+b)

```

## 9. Creación de archivos

El manejo de archivos nos permite enviar información a documentos externos donde se almacena. Para el manejo de archivos es importante llamar la biblioteca estándar <stdio.h> que nos permite usar varias funciones para la edición de archivos.

De forma general debes considerar las siguientes lineas:

1. Agregar un tipo FILE \*Nom\_apuntador
2. Abrir el archivo con `fopen("nombre_archivo.extension", "parametros")`  
`nombre_archivo.extension`, debes darle un nombre al archivo.  
 parámetros puede ser:
  - a) r para leer
  - b) w para escribir
  - c) a para escribir (crear) al final del archivo
  - d) r+ abre un archivo para lectura y escritura de un archivo existente
  - e) w+ crea o sobrescribe (en) un archivo para lectura y escritura
 Hay otros parámetros que no discutiremos por el momento.
3. Cerrar el archivo con `fclose (Nom_apuntador);`
4. Escribir algo en el archivo: Podemos realizarlo de diferentes formas:  
`fputc(variables_introducir, Nom_archi_apuntador)` para poner caracteres en el archivo.

## 10. Estructuras

Una estructura es un tipo de dato que sirve para mantener datos almacenados por el usuario.

Son buenas herramientas creadas para almacenar datos de diferente tipo y mantenerlos en una bitácora o almacén de datos.

La forma general es:

```
struct [etiqueta_de_estructura] {
TIPO definicionA[longitud];
TIPO definicionB[longitud];
TIPO definicionC[longitud];
\dots;
} [one or more structure variables];
```

Ejemplo:

```
struct Books{
    char title[50];
    char author[50];
    char subject[100];
    int book_id; //optional
} book;
```

### Ejemplo 10.1. Un ejemplo de una estructura.

```
1 /* ***** TEXTO SIN ACENTOS ***** */
2 // Recuerda documentar tus codigos
3
4
5 struct corredor
6 {
7     char nombre[30];
8     char sexo;
9     int edad;
10    char categoria[10];
11    float tiempo;
12 };
13
14 // La declaracion de las variables de estructura
15
16 struct corredor c1, c2, c3
```

## 11. Análisis de tiempos en un código

El tiempo de ejecución de un programa depende de:

- El número de procesadores
- La velocidad de lectura/escritura de la memoria
- la arquitectura de la computadora: 32 o 64 bits.
- Los datos de entrada (tamaño)

Para la realización del siguiente análisis se debe considerar:

- se requiere **una** unidad de tiempo para operaciones lógicas o aritméticas.
- se requiere **una** unidad de tiempo para asignar y regresar datos.

Código	Costo	Número de veces
suma (x, y){ return x + y; }	2	1
Tiempo total	2	unidades de tiempo.

**Ejemplo 11.1.** Consideremos una función de la forma:

Para la línea de regreso se requieren **dos** unidades de tiempo.

**Ejemplo 11.2.** Veamos otro ejemplo.

Código	Costo	Número de veces
SumaDeArreglo (X, n){ total = 0; for(i =1; i <= n; i++) total = total + X[i]; return total; }	1 2 2 1	1 n n 1
Tiempo total	4 n + 2	unidades de tiempo.

**Ejemplo 11.3.** Consideremos otro ejemplo.

Código	Costo	Número de veces
SumaDeMatriz (X[], n, m){ total = 0; for(i =1; i <= n; i++) for(j =1; j <= m; j++) total = total + X[i][j]; return total; }	1 2 2 2 1	1 n n×m n×m 1
Tiempo total	4 n×m + 2 n +2	unidades de tiempo.

Considerando los ejemplos anteriores, podemos:

1.  $n \rightarrow \infty$
2.  $n \rightarrow 0$

Existe la notación “Big O”. Nos permite expresar el orden del tiempo de que toma un algoritmo, por ejemplo:  $5n + 4 = O(n)$ .

En esta notación: el tiempo de ejecución es igual a la suma total de todos los fragmentos del código. En general:

- Las declaraciones (definiciones) requieren:  $O(1)$ .
- Las declaraciones iterativas simples requieren:  $O(n)$ .
- Las declaraciones iterativas anidadas requieren:  $O(n \times m)$ .

### 11.1. Estructuras de control

Hemos visto diferentes **estructuras de control**:

- **selectivas (condicionales):**
  1. la sentencia if (y la cláusula else)
  2. El operador ternario
  3. la sentencia switch



■ **repetitivas (iterativas):**

1. bucle while
2. bucle for
3. bucle do-while

## 11.2. Teorema de Böhm y Jacopini

En general...

cualquier programa de ordenador puede diseñarse e implementarse utilizando únicamente las estructuras de control: **secuencia, selección e iteración**.

donde,

**Secuencia:** Conjunto de sentencias que se ejecutan en orden.

**Selección:** Elige cuáles sentencias se ejecutan en función de una condición.

**Iteración:** Las estructuras de control repetitivas repiten conjuntos de instrucciones.

En las construcciones estructurales tenemos secuencias de control donde conocemos el número de iteración (for) y otras que se ejecutan mientras se respeta una condición (while). En este tipo de construcciones estructurales deben un operando con el que se hace válida la iteración.

Es importante entender el problema (usar pseudocódigo, DF). Procure implementar varias formas de solución.

Repasemos algunas ideas del bucle for.

for(i=inicial; i<final; i++)

```

1 #include <stdio.h>
2
3 int main()
4 {
5
6   int inicial = 0, final=10, i;
7
8   for(i=inicial; i<final; i++)
9   {
10    printf("%d\n",i);
11  }
12 }
13
14 /*
15 Nota que tenemos "final-inicial" numero de iteraciones.
16 */

```

for(i=inicial; i<=final; i++)

```

1 #include <stdio.h>
2
3 int main()
4 {
5
6   int inicial = 0, final=10, i;
7
8   for(i=inicial; i<=final; i++)
9   {
10    printf("%d\n",i);
11  }
12 }
13
14 /*
15

```

```
16 Nota que "final+1" numero de iteraciones.
```

```
17
```

```
18 */
```

```
    for(i=inicial; i>final; i--)
```

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

```
2
```

```
3 int main()
```

```
4 {
```

```
5
```

```
6  int inicial = 6, final=10, i;
```

```
7
```

```
8  for(i=final; i>inicial; i--)
```

```
9  {
```

```
10 printf("%d\n",i);
```

```
11 }
```

```
12 }
```

```
13
```

```
14 /*
```

```
15
```

```
16 Nota que "final-inicial" numero de iteraciones.
```

```
17 10
```

```
18 9
```

```
19 8
```

```
20 7
```

```
21 */
```

```
    for(i=inicial; i>=final; i--)
```

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

```
2
```

```
3 int main()
```

```
4 {
```

```
5
```

```
6  int inicial = 4, final=10, i;
```

```
7
```

```
8  for(i=final; i>=inicial; i--)
```

```
9  {
```

```
10 printf("%d\n",i);
```

```
11 }
```

```
12 }
```

```
13
```

```
14 /*
```

```
15
```

```
16 Nota que "final - inicial + 1" numero de iteraciones.
```

```
17 10
```

```
18 9...
```

```
19 4
```

```
20 */
```

```
    for(i=inicial; i<final; i+=k)
```

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

```
2
```

```
3 int main()
```

```
4 {
```

```
5
```

```
6  int inicial = 1, final=12, i,k=2;
```

```
7
```

```
8  for(i=inicial; i<final; i+=k)
```

```
9  {
```

```
10 printf("%d\n",i);
```

```
11 }
```

```
12 }
```

```
13
```

```

14 /*
15
16 Nota que (Final - inicial)/k es el numero de iteraciones.
17 */

    for(i=inicial; i<final; i*=k)

1 #include <stdio.h>
2
3 int main()
4 {
5
6     int inicial = 1, final=25, i,k=2;
7
8     for(i=inicial; i<final; i*=k)
9     {
10         printf("%d\n",i);
11     }
12 }
13
14 /*
15
16 Nota que Log2((final-inicial)/k) es, aproximadamente, el numero de iteraciones.
17 */

    for(i=inicial; i<final; i/=k)

1 #include <stdio.h>
2
3 int main()
4 {
5
6     int inicial = 6, final=0, i,k=5;
7
8     for(i=inicial; i>final; i/=k)
9     {
10         printf("%d\n",i);
11     }
12 }
13
14 /*
15
16 Nota que Log2(final- inicial)/k + 1 es, aproximadamente, el numero de iteraciones.
17 */

```



## Referencias

- Brian W. Kernighan, R. P. (1999). *The practice of programming*. Addison-Wesley.
- Gianfranco Rossi (auth.), E. P. e., Agostino Dovier. (2010). *A 25-Year perspective on logic programming: Achievements of the italian association for logic programming, gulp* (1.<sup>a</sup> ed.). Springer-Verlag Berlin Heidelberg.
- Gottfried, B. S. (1996). *Schaum's outline of theory and problems of programming with c* (2nd Edition ed.). McGraw-Hill Professional.
- Jones, C. (2013). *The technical and social history of software engineering* (1st ed.). Addison-Wesley Professional.
- Juganaru Mathieu, M. (2012). *Introducción a la programación*. Grupo editorial Patria.
- Kochan, S. G. (2004). *Programming in c: A complete introduction to the c programming language*. Developer's Library, Sams.
- L., J. A., y Zahonero Martínez, I. (2001). *Programación en c: Metodología, estructura de datos y objetos*. McGraw-Hill.
- Lytton, W. W. (2002). *From computer to brain foundations of computational neuroscience*. New York: Springer.
- Molina-López, J. (2006). *Fundamentos de programación. grado superior*. McGraw-Hill Interamericana de España S.L.
- Monson-Haefel, R. (2009). *97 things every software architect should know*. O'Reilly Media Inc.
- Owen, K., y Metz, S. (2016). *99 bottles of oop: A practical guide to object-oriented design* (.3 ed.). Potato Canyon Software, LLC.
- Paul Deitel, H. D. (2016). *C how to program. with an introduction to c++* (Global Edition ed.). Pearson International.