Logica de Algoritmos- Un enfoque Practico- Programación en Lenguaje C

Book · January 2016

DOI: 10.13140/RG.2.1.2280.9367

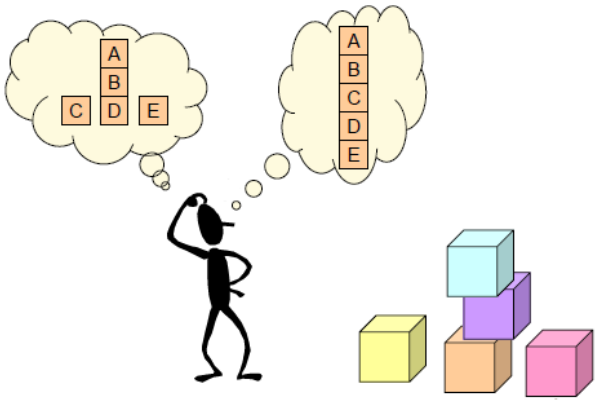
1 author:

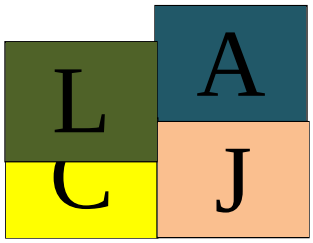
Estevan Gomez
Universidad de las Fuerzas Armadas-ESPE
26 PUBLICATIONS **41** CITATIONS

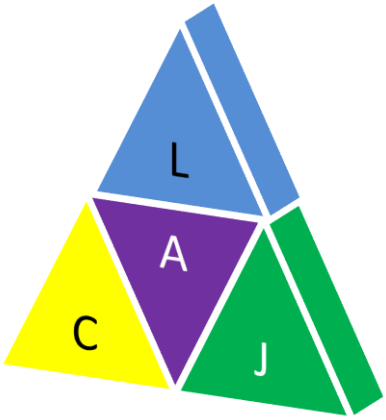
SEE PROFILE

LOGICA DE ALGORITMOS

Introducción a los FUNDAMENTOS DE PROGRAMACION en Lenguaje C







ING ESTEVAN GOMEZ, MSc; estevan.gomez@ute.edu.ec

ING NELSON SALGADO, MSc; nelson.salgado@ute.edu.ec

Tabla de contenidos

Introducción a los FUNDAMENTOS DE PROGRAMACION en Lenguaje C. Primera Edición

ISBN: 978-9942-21-974-9
Enero 2016
Quito – Ecuador

Introducción

El Mundo de la Programación es basto e incluye muchas connotaciones desde lo meramente trivial hasta lo más complejo, actualmente los Algoritmos requieren o están ligados a la Programación de Computadores, Aplicaciones para Dispositivos Móviles, Redes, y sistemas de Información, entre las miles de aplicaciones disponibles en el Internet.

Actualmente existen según tecnomagazine alrededor de 1'000.000 de aplicaciones disponibles en el mercado: "El mercado de los smartphones sigue creciendo y debido a esto **cada vez hay mayor cantidad de aplicaciones** y de descargas, Si bien es un gran número, no todas están activas, es decir, no todas pueden ser descargadas, solamente se encuentra activo un 80% del total, es decir unas 800.000. **Del millón de apps, se calcula que iOS tiene un 52%, mientras que Android se queda con el 48% restante**." **(iOS es** un sistema operativo móvil de la empresa Apple Inc.)

La influencia del código abierto en la participación de programadores a lo ancho y largo del planeta ha creado comunidades extensas de programadores que se contactan virtualmente y trabajan mancomunadamente por objetivos comunes.
Microsoft acaba de lanzar su nuevo Sistema operativo Windows 8, mientras el Smartphone cubre las necesidades de estudiantes y ejecutivos, y el auge de las redes sociales permite la intercomunicación de los más vastos sectores sociales, económicos, y académicos, existe un creciente y siempre demandante sector que requiere nuevas aplicaciones las cuales deben facilitar las actividades del usuario común, como pagar entradas para el cine, explotar las nuevas potencialidades que ofrece el hardware, es decir el algoritmo es la base del software el cual a su vez es el alma que da vida al hardware disponible.

"Hoy en día, nuestros teléfono inteligentes son prácticamente miniordenadores móviles donde llevamos una gran parte de nuestra vida. Ya no solamente los utilizamos para llamar o enviar mensajes, sino así mismo para mirar correos electrónicos, encontrar información, mirar cómo llegar de un lugar a otro ya sea por transporte público, auto o a pie, examinar redes sociales, trabajar, mirar videos y, por probable, escuchar música, desde donde nos encontremos."[1]

Objetivos:

El objetivo principal es que el presente libro exprese de manera clara los conceptos, el análisis requerido para desarrollar un algoritmo y mostrar a través de ejemplos de Algoritmos, su fundamentación, conceptualización y creación de manera práctica, para que sea utilizado por : estudiantes, catedráticos y todas aquellas personas que se inician en la programación.

[1] http://www.pysnnoticias.com/2013/04/07/mejores-aplicaciones-sistema-android-para-descargar-music-2/

Este libro pretende sentar las bases para que aquellos que se inician en el fascinante campo de la programación lo utilicen como un manual de consulta.

¿Por qué leer este libro?

Haz visto mi agenda le decía el jefe a su secretaria en 1970, en el 2015 el mismo jefe le pregunta a su secretaria, me enviaste la agenda del mes a mi e-mail, revisándolo es su Smartphone. "Como han cambiado los tiempos dice la mayoría de la gente, esto obedece al uso cada vez más extenso de la tecnología, a través de la incursión de las redes sociales en el Smartphone o la tablet".Preparémonos entonces para ingresar al innovativo mundo de la programación

¿Qué beneficios le puede dar el leer este libro?

A pesar de que existen varios libros sobre al respecto, lo que se pretende es tratar el tema de una forma práctica, con una visión actual, pero sobre todo clara, está concebido como una herramienta de consulta para todo aquel que esté relacionado con los inicios en programación.

Este libro presenta de una manera fácil y entendible los conceptos, mezclándolas con explicaciones y analogías que permiten al lector inferir conceptos y sus aplicaciones de forma paulatina y sistemática, es decir partiendo de conceptos básicos se llegará a plantear algoritmos y programas medianamente complejos y el lector podrá plantear una serie de algoritmos de acuerdo a su necesidad.

Contribuciones de éste trabajo:

Las contribuciones de éste trabajo son: Expresar un modelo de inmersión en estudio de la problemática de desarrollo de programas basados en la lógica de construcción de algoritmos de forma práctica, partiendo de la observación, pasando al análisis y finalmente a la construcción y desarrollo de algoritmos. Cada capítulo se desarrolla en base a objetivos establecidos, describiendo la fundamentación teórica necesaria , permitiendo la comprensión a través de ejercicios propuestos y resueltos y la recopilación de conceptos clave de capitulo a fin de lograr mejor comprensión de los temas tratados.

Este libro se estructura en: Capítulo 1 en donde se cubre la base teórica fundamental de algoritmos, con diversas fundamentaciones teóricas y prácticas , así como desarrollando varios ejemplos que permitan la comprensión del capítulo, Capitulo2 se definen programas, su conceptualización y se da una perspectiva de Programación, Capitulo 3: se desarrolla las características básicas de Lenguaje C, Capitulo 4: Tipos de Ciclos Aplicables a Lenguaje C,Capitulo 5: Funciones en C, Capitulo 6: Se desarrolla Arreglos Unidimensionales y Bidimensionales.

Blog:

Como soporte adicional y de consulta se ha desarrollado el Blog Fundamentos de Programación: https://sites.google.com/a/espe.edu.ec/fundamentos-de-programacion/

Capítulo No 1:

1.1. Conceptualización:

Para poder desarrollar algoritmos, lo primero que debemos hacer es lograr entender los conceptos, es decir conceptualizar, según(WordReference, 2015) la conceptualización se conoce como: "Elaboración detallada y organizada de un concepto a partir de datos concretos o reales."

Entonces: ¿cómo conceptualizamos de inicio?
Qué es el teclado virtual? Le pregunta la Sra. a su hijo, entonces, él le responde, mira cuando tú necesitas enviar un email, haces clic aquí y te sale un teclado, entonces empiezas a escribir.
Para Ud. qué es el teclado virtual?
R: Un algoritmo, Un programa, una aplicación?
En este libro pretendemos que los Fundamentos de Programación, a través de la Teoría de Algoritmos sean tratados de la manera más natural, a fin de que sea útil para toda persona que empiece en este trabajo, que conozca del tema, o que simplemente esté interesada en conocer más.

1.2. Algoritmo:

Haz cambiado una llanta de tu vehículo?,Haz preparado una torta de huevo?, Ok entonces ya podemos empezar

"En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, unalgoritmo (del griego y latín, dixit algorithmus y este a su vez del matemático persa Al-Juarismi) es un conjunto preescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad , dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución."2

Concepto Clave: En otras palabras: "El algoritmo me permite a través de una serie de pasos lograr la solución de un determinado problema, tomando en cuenta las condiciones internas y externas que lo afectan o condicionan, si cada vez se sigue esta secuencia de pasos debe lograse la misma solución.

1.2.1. Definición Formal de Algoritmo:

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo.
Muchos autores los señalan como listas de instrucciones para resolver un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema

[2] http://es.wikipedia.org/wiki/Algoritmo

(entrada) en una solución (salida). Sin embargo cabe notar que algunos algoritmos no necesariamente tienen que terminar o resolver un problema en particular.

Por ejemplo, una versión modificada de la "criba de Eratóstenes" que nunca termine de calcular números primos no deja de ser un algoritmo.

A lo largo de la historia varios autores han tratado de definir formalmente a los algoritmos utilizando modelos matemáticos como máquinas de Turing entre otros Sin embargo, estos modelos están sujetos a un tipo particular de datos como son números, símbolos o gráficas mientras que, en general, los algoritmos funcionan sobre una vasta cantidad de estructuras de datos. En general, la parte común en todas las definiciones se puede resumir en las siguientes tres propiedades siempre y cuando no consideremos algoritmos paralelos:

Tiempo secuencial. Un algoritmo funciona en tiempo discretizado –paso a paso–, definiendo así una secuencia de estados "computacionales" por cada entrada válida (la entrada son los datos que se le suministran al algoritmo antes de comenzar).

Estado abstracto. Cada estado computacional puede ser descrito formalmente utilizando una Estructura de primer orden y cada algoritmo es independiente de su implementación (los algoritmos son objetos abstractos) de manera que en un algoritmo las estructuras de primer orden son invariantes bajo isomorfismo.

Exploración acotada. La transición de un estado al siguiente queda completamente determinada por una descripción fija y finita; es decir, entre cada estado y el siguiente solamente se puede tomar en cuenta una cantidad fija y limitada de términos del estado actual

Podemos decir que Algoritmo es una serie de pasos sucesivos, con el objetivo de dar solución a un problema, y cuya característica es tener un inicio y un fin.

1.2.2. Ejemplos:

Veamos el ejemplo de la tortilla de huevo:

Podemos escribir lo que realizaremos en palabras sencillas: "Escojo el huevo, pico el tomate y la cebolla, coloco los ingredientes en el tazón, agrego un poco de sal y los bato hasta que salga una espuma, preparo el sartén y lo pongo al fuego, vierto el aceite en el sartén, y caliento el sartén hasta que el aceite emita una pequeña humareda, finalmente espero hasta que se esponje y doy la vuelta a la tortilla con una paleta"

Esa es la versión más común de cómo prepara una tortilla de huevo

¿Ahora como expresamos este algoritmo? , ¿Qué necesitamos para que sea comprendido?, lo vamos a ver más adelante.

Concepto Clave: En resumen: un algoritmo es cualquier procedimiento que funcione paso a paso, donde cada paso se pueda describir sin ambigüedad y sin hacer referencia a una computadora en particular, y además tiene un límite fijo en cuanto a la cantidad de datos que se pueden leer/escribir en un solo paso

1.2.3. Características de un Algoritmo:

1. **Entrada**: definir lo que necesita el algoritmo
2. **Salida**: definir lo que produce.
3. **No ambiguo**: explícito, siempre sabe qué comando ejecutar.
4. **Finito**: El algoritmo termina en un número finito de pasos.
5. **Correcto**: Hace lo que se supone que debe hacer. La solución es correcta
6. **Efectivo**: Cada instrucción se completa en tiempo finito. Cada instrucción debe ser lo suficientemente básica como para que en principio pueda ser ejecutada por cualquier persona usando papel y lápiz.
7. **Preciso**: implica el orden de realización de cada uno de los pasos
8. **Definido:** si se sigue dos veces, se obtiene el mismo resultado.

1.2.4. Tipos de Algoritmos:

Método	Descripción	Ejemplos
Algorítmico	Utiliza un algoritmo y puede ser implementado en una computadora	Instrucciones para manejar un vehículo Instrucciones para secar grano a granel Instrucciones para resolver ecuación de segundo grado
Heurística:	Se apoya en el resultado obtenido en un análisis de alternativas de experiencias anteriores similares. De las mismas, a se deducen una serie de reglas empíricas o heurísticas que de ser seguidas, conducen a laselección de la mejor alternativa en todas o la mayoría de las veces.	Utilización de la heurística para la elaboración de sistemas expertos La capacidad heurística es un rasgo característico de los humanos, desde cuyo punto de vista puede describirse como *el arte y la ciencia del descubrimiento y de la invención* o de resolver problemas mediante la creatividad y el pensamiento lateral o pensamiento divergente.

Los algoritmos se pueden expresar por:
1.-Formulas

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

2.- Diagramas de flujo

3.- Pseudo código

1.3. Medios de expresión de un algoritmo

Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo ylenguajes de programación entre otros.

- Las descripciones en lenguaje natural tienden a ser ambiguas y extensas.
- El usar pseudocódigo y diagramas de flujo evita muchas ambigüedades del lenguaje natural.
- Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico.

La descripción de un algoritmo usualmente se hace en tres niveles:

1. **Descripción de alto nivel**. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles.
2. **Descripción formal**. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.
3. **Implementación**. Se muestra el algoritmo expresado en un lenguaje de programación específico o algún objeto capaz de llevar a cabo instrucciones.

También es posible incluir un teorema que demuestre que el algoritmo es correcto, un análisis de complejidad o ambos.

1.3.1. Descripción Narrada (Lenguaje Natural)

Este algoritmo es caracterizado porque sigue un proceso de ejecución común y lógico, describiendo textualmente paso a paso cada una de las actividades a realizar dentro de una actividad determinada.

[3]Ejemplo: Algoritmo para asistir a clases:

1. Levantarse
2. Bañarse
3. Vestirse
4. Desayunar
5. Cepillarse los dientes
6. Salir de casa
7. Tomar el autobús
8. Llegar a la Universidad
9. Buscar el aula
10. Ubicarse en un asiento

Ejemplo: Algoritmo tomar una ducha:

1.-Desvestirse
2.-Ajustar la temperatura del agua
3.-Mojar todo el cuerpo
4.-Jabonarse el cuerpo y colocarse shampoo
5.-Lavar tu rostro
6.- Lavar tu cuerpo
7.-Enjuagar el Jabòn
8.- Secarse el cuerpo con una toalla
9.- Fin

[3] https://www.google.com.ec/imgres?imgurl=http://pintarimagenes.org/wp-content/uploads/2015/02/ni%2525C3%2525B1os-camino-a-la-escuela.png&imgrefurl=http://pintarimagenes.org/dibujos-de-ninos-camino-a-la-escuela-para-pintar/&h=450&w=600&tbnid=lkyCSkp1MeRulM:&docid=xY8XMUPBkVgBJM&ei=kaqTVvSCB8mze8mNqLAH&tbm=isch&ved=0ahUKEwi0j5Or66HKAhXJ2R4KHckGCnYQMwhdKCMwIw&biw=1024&bih=623

1.3.2. Pseudocódigo

El pseudocódigo (falso lenguaje, el prefijo pseudo significa falso) es una descripción de alto nivel de un algoritmo que emplea una mezcla de lenguaje natural con algunas convenciones sintácticas propias de lenguajes de programación, como asignaciones, ciclos y condicionales, aunque no está regido por ningún estándar. Es utilizado para describir algoritmos en libros y publicaciones científicas, y como producto intermedio durante el desarrollo de un algoritmo, como los Diagramas de flujo, aunque presentan una ventaja importante sobre estos, y es que los algoritmos descritos en pseudocódigo requieren menos espacio para representar instrucciones complejas.

El pseudocódigo está pensado para facilitar a las personas el entendimiento de un algoritmo, y por lo tanto puede omitir detalles irrelevantes que son necesarios en una implementación. Programadores diferentes suelen utilizar convenciones distintas, que pueden estar basadas en la sintaxis de lenguajes de programación concretos. Sin embargo, el pseudocódigo, en general, es comprensible sin necesidad de conocer o utilizar un entorno de programación específico, y es a la vez suficientemente estructurado para que su implementación se pueda hacer directamente a partir de él.

Ejemplo:

Diseñar un algoritmo que lea cuatro variables y calcule e imprima su producto, suma y media aritmética.

```
inicio
leer (a, b, c, d)
producto <-- (a * b * c * d)
suma <-- (a + b + c + d)
media <-- (a + b + c + d) / 4
escribir (producto, suma, media)
fin
```

Concepto Clave: Así el pseudodocódigo cumple con las funciones antes mencionadas para representar algo abstracto los protocolos son los lenguajes para la programación. Busque fuentes más precisas para tener mayor comprensión del tema.

1.3.3. Diagramas N-S

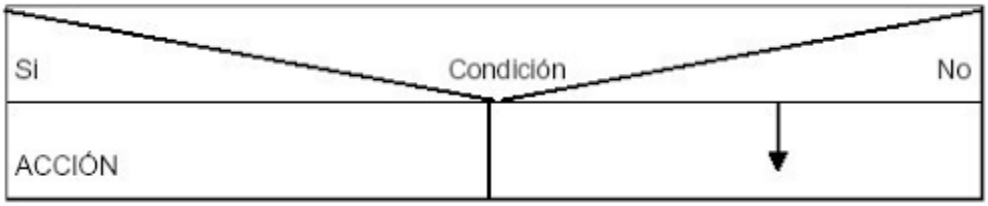
Son una herramienta que favorece la programación estructurada y reúne características gráficas propias de diagramas de flujo y lingüísticas propias de pseudocódigos. Constan de una serie de cajas contiguas que se leerán siempre de arriba-abajo y sus estructuras lógicas son las siguientes:

[4] https://www.google.com.ec/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwizjPCB0qDKAhXDlB4KHX8UCYEQjRwIBw&url=http%3A%2F%2Fgrupos.emagister.com%2Fdebate%2Fpara_sil_codol%2F2375-756635&psig=AFQjCNGcEYeZGhFdS4B0wTppaeR_-lX43g&ust=1452563229648390

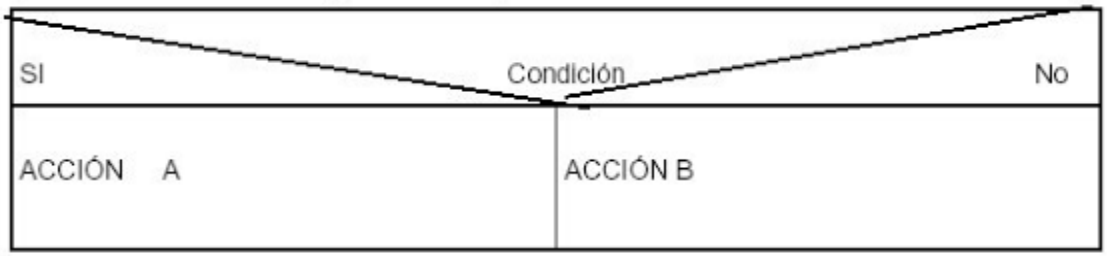
Estructura Secuencial

Acción A
Acción B
.....
.....
Acción N

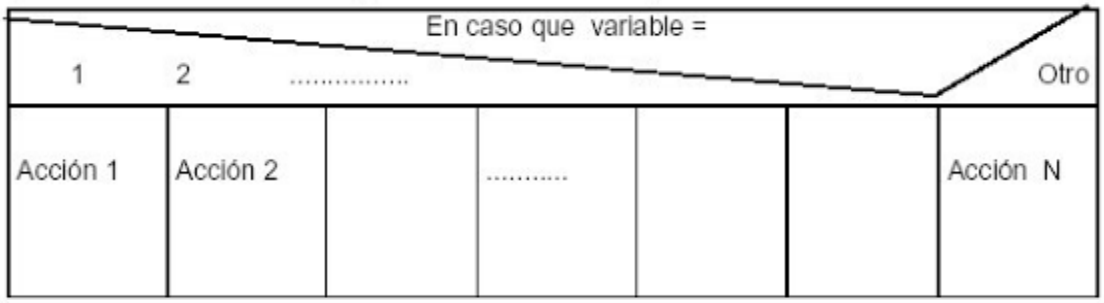
Estructura Selectiva Simple (If-Then)



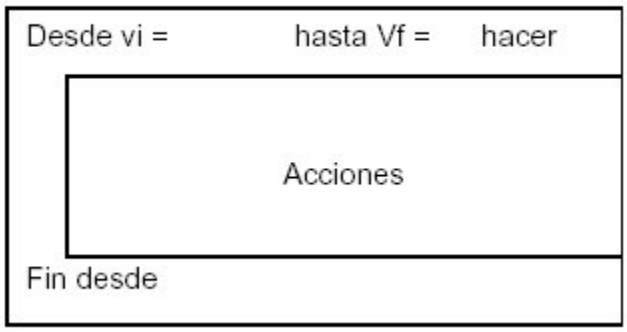
Estructura Selectiva Doble (If-Then-Else)



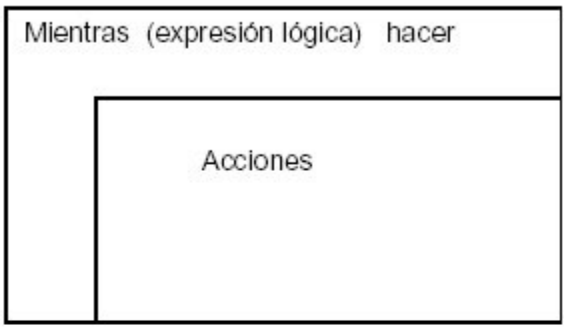
Estructura Selectiva Múltiple (Select- Case Endcase)



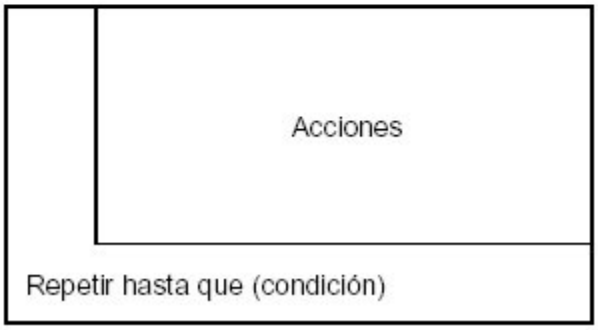
Estructuras Repetitivas (Desde/Para)



Estructura Iterativa Mientras



Estructura Iterativa Repetir



Por ejemplo, el ejercicio anterior se puede representar así:

Inicio
Leer (a,b,c,d)
Producto ← (a*b*c*d)
Suma ← (a+b+c+d)
Media ← suma/4
Escribir (producto, suma, media)
Fin

1.3.4. Diagrama de flujo

Los diagramas de flujo son descripciones gráficas de algoritmos; usan símbolos conectados con flechas para indicar la secuencia de instrucciones y están regidos por ISO.[5]

Los diagramas de flujo son usados para representar algoritmos pequeños, ya que abarcan mucho espacio y su construcción es laboriosa. Por su facilidad de lectura son usados como introducción a los algoritmos, descripción de un lenguaje y descripción de procesos a personas ajenas a la computación.

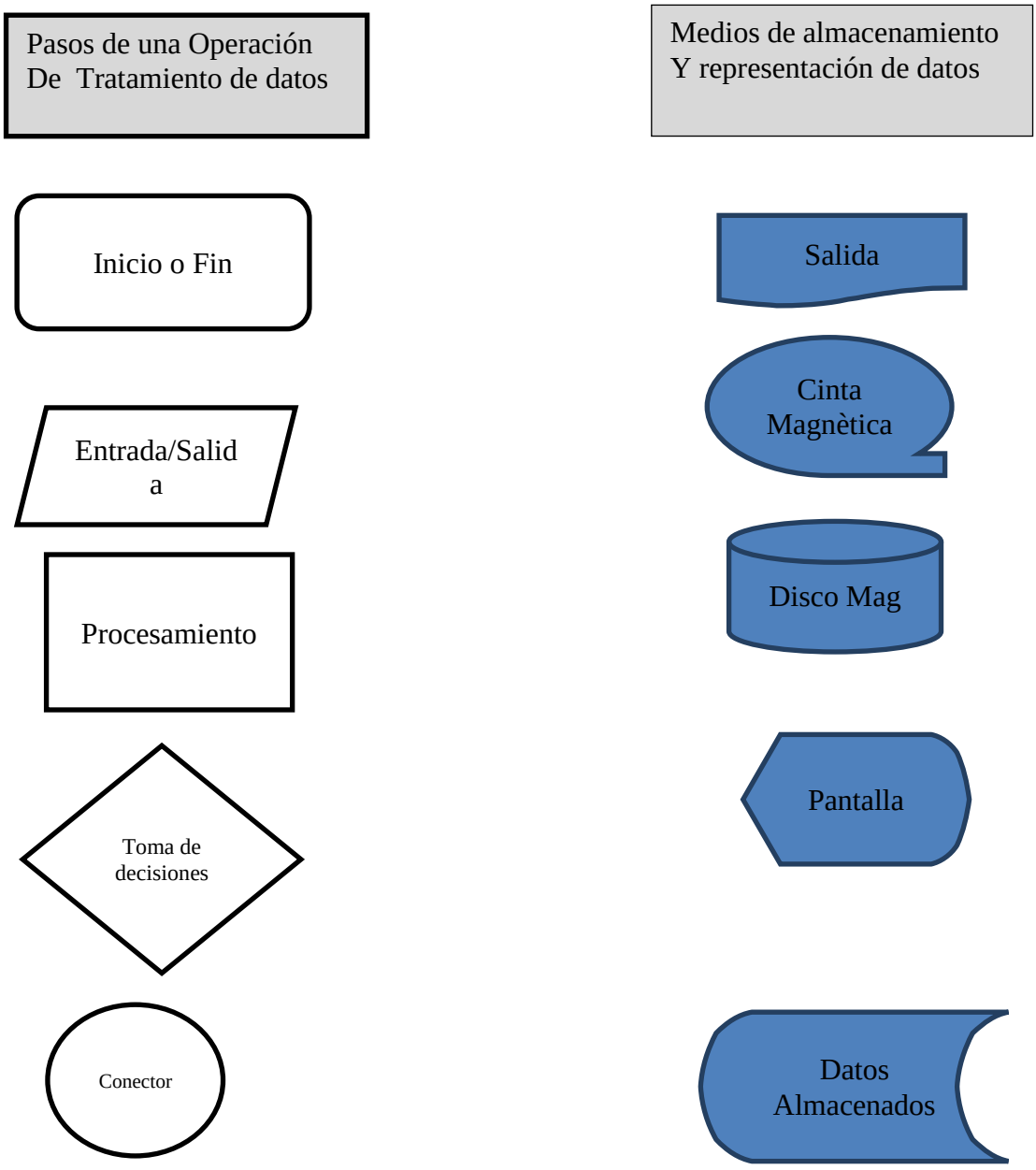
Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación entre otros. Las descripciones en lenguaje natural tienden a ser ambiguas y extensas. El usar pseudocódigo y diagramas de flujo evita muchas ambigüedades del lenguaje natural. Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico.

1.3.5. Reglas para la elaboración de Diagramas de Flujo:

1. Se escribe de arriba hacia abajo y de izquierda a derecha
2. Siempre se usan flechas verticales u horizontales, jamás curvas
3. Evitar cruce de flujos
4. En cada paso expresar una acción concreta
5. Secuencia de flujo normal en una solución de problema
6. Tiene un inicio
7. Una lectura o entrada de datos
8. El proceso de datos
9. Una salida de información
10. Un final

[5] http://www.iso.org/iso/home.html

1.3.6. Notación de Algoritmos:



1.3.7. VENTAJAS DE USAR FLUJOGRAMAS

Rápida comprensión de las relaciones
Análisis efectivo de las diferentes secciones del programa
Pueden usarse como modelos de trabajo en el diseño de nuevos programas o sistemas
Comunicación con el usuario
Documentación adecuada de los programas
Codificación eficaz de los programas
Depuración y pruebas ordenadas de programas

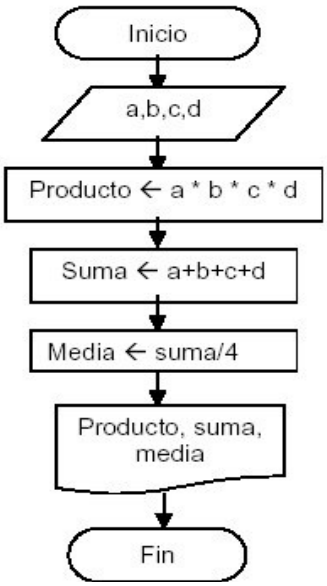
1.3.8. DESVENTAJAS DE LOS FLUJOGRAMAS

Diagramas complejos y detallados suelen ser laboriosos en su planteamiento y diseño
Acciones a seguir tras la salida de un símbolo de decisión, pueden ser difíciles de seguir si existen diferentes caminos
No existen normas fijas para la elaboración de los diagramas de flujo que permitan incluir todos los detalles que el usuario desee introducir.
Representando el ejemplo como flujograma tenemos:

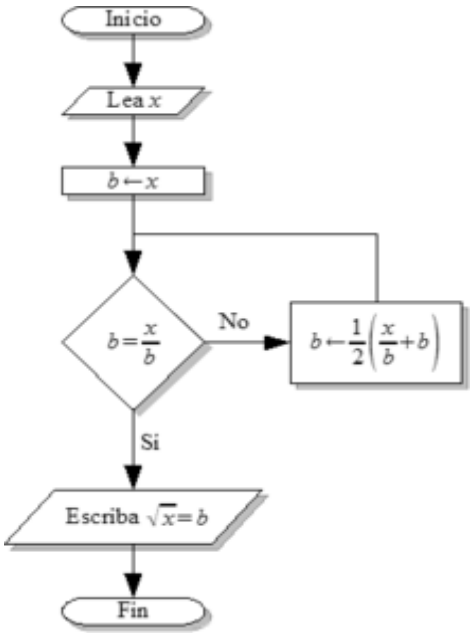
Concepto Clave:

- Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación entre otros
- El pseudocòdigo es una expresión paso a paso de una solución.

EJEMPLO 1 :

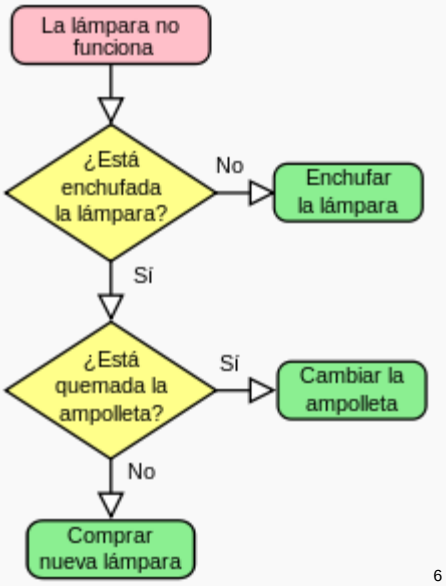


EJEMPLO 2 :



EJEMPLO 3:

Diagrama de flujo que expresa un algoritmo para calcular la raíz cuadrada de un número



[6]

Diagrama de flujo sencillo con los pasos a seguir si una lámpara no funciona.

El **diagrama de flujo** o **diagrama de actividades** es la representación gráfica delalgoritmo o proceso. Se utiliza en disciplinas como programación, economía, procesos industriales y psicología cognitiva.

En Lenguaje Unificado de Modelado (UML), un diagrama de actividades representa los flujos de trabajo paso a paso de negocio y operacionales de los componentes en un sistema. Un diagrama de actividades muestra el flujo de control general.

En SysML el diagrama de actividades ha sido extendido para indicar flujos entre pasos que mueven elementos físicos (e.g., gasolina) o energía (e.g., presión). Los cambios adicionales permiten al diagrama soportar mejor flujos de comportamiento y datos continuos.

Estos diagramas utilizan símbolos con significados definidos que representan los pasos del algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y de fin de proceso.

1.3.9. Características de un Algoritmo:

Un diagrama de flujo siempre tiene un único punto de inicio y un único punto de término.

Las siguientes son acciones previas a la realización del diagrama de flujo:

[6] https://www.google.com.ec/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjYjrimu6DKAhXG7B4KHaVrDHYQjRwIBw&url=http%3A%2F%2Fwordpress.danieltubau.com%2Fel-algoritmo-de-polifemo%2Falgoritmo-lampara-svg%2F&psig=AFQjCNF1pRph2epro4TkMF_A6r6MVn-OIw&ust=1452557172651242

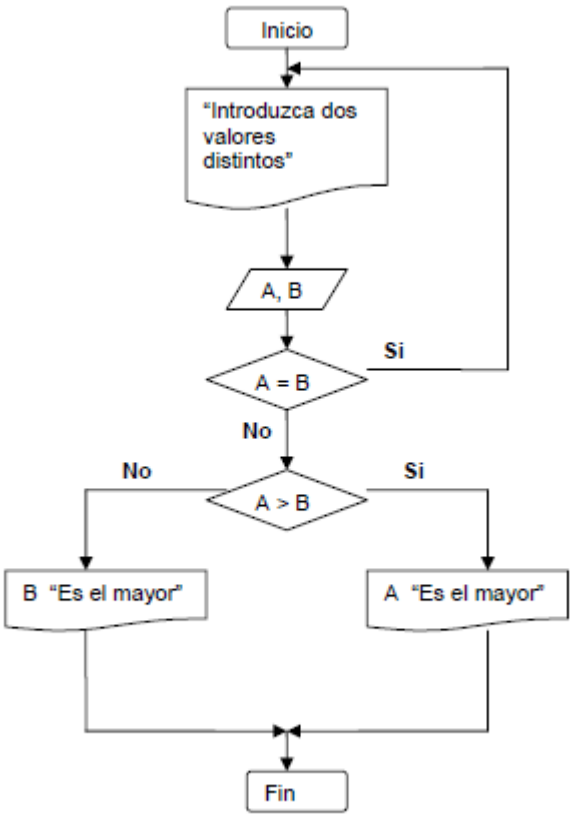
- Identificar las ideas principales a ser incluidas en el diagrama de flujo. Deben estar presentes el autor o responsable del proceso, los autores o responsables del proceso anterior y posterior y de otros procesos interrelacionados, así como las terceras partes interesadas.
- Definir qué se espera obtener del diagrama de flujo.
- Identificar quién lo empleará y cómo.
- Establecer el nivel de detalle requerido.
- Determinar los límites del proceso a describir.

Los pasos a seguir para construir el diagrama de flujo son:

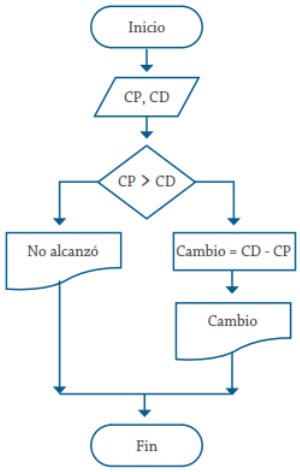
- Establecer el alcance del proceso a describir. De esta manera quedará fijado el comienzo y el final del diagrama. Frecuentemente el comienzo es la salida del proceso previo y el final la entrada al proceso siguiente.
- Identificar y listar las principales actividades/subprocesos que están incluidos en el proceso a describir y su orden cronológico.
- Si el nivel de detalle definido incluye actividades menores, listarlas también.
- Identificar y listar los puntos de decisión.
- Construir el diagrama respetando la secuencia cronológica y asignando los correspondientes símbolos.
- Asignar un título al diagrama y verifica

EJERCICIOS RESUELTOS

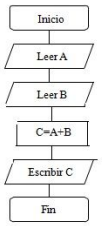
Ejercicio No 1: Desarrolle un algoritmo que permita leer dos valores distintos, determinar cuál de los dos valores es el mayor y escribirlo

Pseudocódigo	Diagrama de Flujo
1. Inicio 2. Inicializar variables: A = 0, B = 0 3. Solicitar la introducción de dos valores distintos 4. **Leer** los dos valores 5. Asignarlos a las variables A y B 6. **Si** A = B **Entonces** vuelve a 3 porque los valores deben ser distintos **7. Si** A>B **Entonces** **Escribir** A, "Es el mayor" 8. **De lo contrario**: **Escribir** B, "Es el mayor" **9. Fin_Si** **10. Fin**	

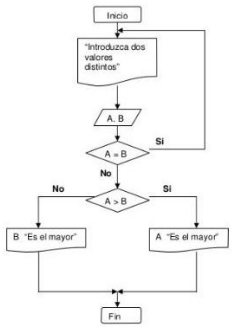
Ejercicio 2: Se requiere un algoritmo para determinar el cambio que recibirá una persona que adquiere un producto en la tienda.

Pseudocódigo	Diagrama de Flujo
1. Inicio 2. Inicializar variables: CP = 0,CD = 0 3. **Leer** los dos valores 5. Asignarlos a las variables CD y CP 6. **Si CP> CDEntonces** **7. CAMBIO= CD-CP** **8. Escribir** CAMBIO **9. De lo contrario**: **Escribir** , "No alcanzò" **10. Fin_Si** **11. Fin**	

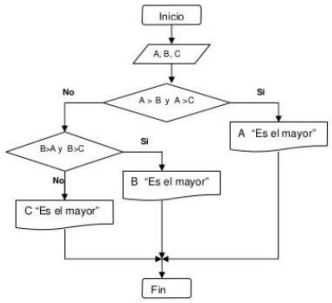
Ejercicio 3: Hacer el diagrama de flujo para sumar dos números leídos por teclado y escribir el resultado**.**

Pseudocódigo	Diagrama de Flujo
→ *Leemos el primer número y lo dejamos en A* → *Leemos el segundo número y lo dejamos en B* → *Sumamos A y B, y dejamos el resultado en C* → *Escribimos C*	

Ejercicio 4: Hacer un diagrama de flujo que permita leer 2 números diferentes y nos diga cuál es el mayor de los 2 números.

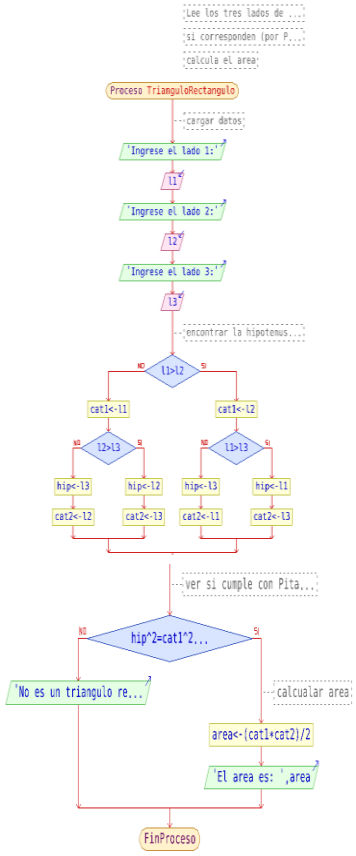
Pseudocódigo	Diagrama de Flujo
1.Inicio 2.Inicializar variables: A = 0, B = 0 3.Solicitar la introducción de dos valores distintos 4.**Leer** los dos valores 5.Asignarlos a las variables A y B 6.**Si** A = B **Entonces** vuelve a 3 porque los valores deben ser distintos **7.Si** A>B **Entonces** **Escribir** A, "Es el mayor" 8.**De lo contrario**: **Escribir** B, "Es el mayor" **9.Fin_Si** **10. Fin**	

Ejercicio 5: Crear un diagrama de flujo de procesos en el que se almacenen 3 números en 3 variables A, B y C. El diagrama debe decidir cuál es el mayor y cuál es el menor.

Pseudocódigo	Diagrama de Flujo
leer a; leer b; leer c; si a> b y b> c entonces Escribir "A es el mayor" si b>a y b>c Escribir "B es el mayor" Sino Escribir "C es el mayor" FinSi FinSi	

Ejercicio 6: Lee los tres lados de un triangulo rectangulo, determina si corresponden (por Pitargoras) y en caso afirmativo , calcula el area

Pseudocódigo	Diagrama de Flujo

Escribir "Ingrese el lado 1:"
Leer l1
Escribir "Ingrese el lado 2:"
Leer l2
Escribir "Ingrese el lado 3:"
Leer l3
// encontrar la hipotenusa (mayor lado)
Si l1>l2 Entonces
cat1<-l2
Si l1>l3 Entonces
hip<-l1
cat2<-l3
Sino
hip<-l3
cat2<-l1
FinSi
Sino
cat1<-l1
Si l2>l3 Entonces
hip<-l2
cat2<-l3
Sino
hip<-l3
cat2<-l2
FinSi
FinSi
 	
// ver si cumple con Pitagoras
Si hip^2 = cat1^2 + cat2^2 Entonces
// calcualar area
area<-(cat1*cat2)/2
Escribir "El area es: ",area
Sino
Escribir "No es un triángulo rectangulo."
FinSi
	

1.3.10. Ejercicios Resueltos de Pseudocódigo

1.- Escribir un Pseudocódigo de un programa que permita leer la edad y peso de una persona y posteriormente imprimirla.

```
Inicio
Variables edad, peso.
Imprimir "Escribir los datos (Edad, Peso):"
Leer Edad, Leer Peso.
Visualizar "Tu peso es: ", peso, " y tu edad es: ", edad.
Fin.
```

2.- Escribir un Pseudocódigo que calcule el área de un triángulo recibiendo como entrada el valor de base y altura.

```
Inicio
Variables Altura, Base, Area.
Imprimir "Introduce la base y la altura: "
Leer base y altura.
Area= (base*altura)/2.
Imprimir "El area es: ", Area
Fin.
```

3.- Escribir Pseudocódigo que calcule el área de un círculo.

```
Inicio
Constantes Pi= 3.1416
Variables Radio, area = real
Imprime "Introduce el radio: "
Leer radio.
area= radio*radio*Pi
Imprimir "El área del circulo es: ", area.
Fin.
```

4.- Escribir Pseudocodigo que dados 2 valores de entrada imprima siempre la división del mayor entre el menor.

```
Inicio
Variables num1, num2=entero.
Variables R=real.
Imprimir "Introduce los números:"
Leer num1, Leer num2.
Si a>b entonces
       R= a/b
  Sino
    R= b/a
Finsi
Imprimir "La división es =", R;
```

5.- Escribir Pseudocódigo que lea de entrada 3 números y que indique cual es el mayor de ellos.

```
Inicio
Variables a, b,c = enteros.
Imprimir "Introduce los datos a comparar: "
Leer a,b,c
Si a<b y a<c entonces
      Imprime "El mayor es: ", a
    Sino
      Si b>a y b>c entonces
       Imprime "El mayor es: ",b
       sino
         Imprime "El mayor es: ", c
Fin.
```

6.- Escribir un Pseudocódigo que lea 3 números los cuales significan una fecha (día, mes, año). Comprobar que sea válida la fecha, si no es válido que imprima un mensaje de error, y si es válida imprimir el mes con su nombre.

```
Inicio
Variables dia, mes, año =entero.

Imprimir "Introduce la fecha (Dia,mes,año): "
Leer dia
leer  mes
leer  año.

Si dia>31 o mes >12 o año<0 entonces
Imprimir "Error la fecha no es correcta"
Sino
si mes=1 o mes=3 o mes=5 o mes=7 o mes=8 o mes=10 o mes=12 entonces
si dia >31 y dia<1 entonces
Imprimir "Error de Dia"
sino
si mes=1 entonces imprimir dia,"/ Enero /",año
si mes=3 entonces imprimir dia,"/ Marzo /",año
si mes=5 entonces imprimir dia,"/ Mayo /",año
si mes=7 entonces imprimir dia,"/ Julio /",año
si mes=8 entonces imprimir dia,"/ Agosto /",año
si mes=10 entonces imprimir dia,"/ Octubre /",año
si mes=12 entonces imprimir dia,"/ Diciembre /",año
Si mes=2 entonces
si dia>28 o dia<0 entonces
```

```
Imprimir "Error de Dia"
sino
Imprimir dia,"/ Febrero /",año
 Si mes=4 o mes=6 o mes=9 o mes=11
Si dia >30 o dia <1 entonces
Imprimir "Error de Dia"
Sino
Si mes=4 entonces
Imprimir dia,"/ Abril /",año
Si mes=6 entonces
Imprimir dia,"/ Junio /",año
Si mes=9 entonces
Imprimir dia,"/ Septiembre /",año
Si mes=11 entonces
Imprimir dia,"/ Noviembre /",año
Fin.
```

7.- Escribir un Pseudocodigo que pida la edad y el sexo y dependiendo si es hombre o mujer y si puede votar o no.

```
Inicio
variables edad=entero, sexo=caracter.
repetir
Imprimir "Introduce Tu edad:"
Leer edad.
Hasta que (edad >0)

Hacer
Imprimir "Introduce tu sexo (M/H):"
leer sexo

hasta que (sexo='H' o sexo = 'M')

Si sexo= 'M' entonces
si edad > 18 entonces
Imprimir "Eres Mujer y puedes votar"

sino
Imprimir "Eres Mujer y no puedes votar"

Sino

si edad >18 entonces

Imprimir "Eres Hombre y puedes votar"
```

sino

Imprimir "Eres Hombre y no puedes votar"

Fin.

8.- Realice un Pseudocódigo que calcule la nómina salarial neto, de unos obreros cuyo trabajo se paga en horas. El cálculo se realiza de la siguiente forma:

- Las primeras 35 horas a una tarifa fija.

- Las horas extras se pagan a 1.5 más de la tarifa fija.

- Los impuestos a deducir de los trabajadores varian, segun el sueldo mensual si el sueldo es menos a $20,000.00 el sueldo es libre de impuesto y si es al contrario se cobrará un 20% de impuesto.

Inicio

Canstante Tarifa= 50.

Variables Horas, Sueldo, dif_horas, tarifa_extra, Salario_extra, Sueldo_mes, Impuesto, Sueldo_total.

Imprimir "Introduce las Horas de Trabajo ==> "
Leer Horas.

Si Horas <= 35 entonces

Sueldo= horas * tarifa.

Sino

Dif_horas= Horas - 35

tarifa_extra=(tarifa * 1.5)

Salario_exta= tarifa_extra * Dif_horas

Sueldo=(35*tarifa)+ salario_extra.

Sueldo_mes = sueldo * 4

Impuesto=0

Si sueldo_mes > 20000 entonces

Impuesto = (Sueldo_mes* 0.20)

Salario_total = Sueldo_mes - Impuesto

Imprimir "Tu sueldo al mes es: ", sueldo_mes,"Valor de Impuesto: ", Impuesto, " El salario Neto es: ",Salario_total.

sino

Imprimir "Tu sueldo Neto es: ", sueldo_mes.

Fin.

9.- Escribir un Pseudocódigo que encuentre y despliegue los números primos entre uno y cien. Un número primo es divisible entre el mismo y la unidad por lo tanto un numero primo no puede ser par excepto el dos (2).

Inicio
Variables primo, cont, div, res.
Primo = 1.
Hacer mientras primo<=100
Div =0.
Cont =1.

Hacer Mientras cont <= primo

Res = cont mod primo

si res = 0 entonces

Div = div +1.

Fin si

Cont = cont +1.

Fin de Hacer mientras

si div<=2 entonces

imprimir primo

Fin si.

Primo = primo + 1.

Fin de Hacer mientras

Fin.

10.- Hacer un Pseudocódigo que calcule el factorial de un número.

Inicio

Variables N,Fact,aux.

Imprimir "Introduce el número: "

Leer N

aux= n-1.

fact=n

Hacer
    fact=fact * aux.

  Hasta que aux=1

    Imprimir "El factorial de ", n, "es:", fact

Fin.

11.- Hacer un Pseudocodigo que despliegue las tablas de multiplicar.

Inicio

Variables i,k,r.

para i=1 hasta 10.

para k=1 hasta 10.

r:=i*k.

Imprimir i," por ",k," = ",r.

k=k+1.

i=i+1.

Fin.

EJERCICIOS TIPO 2

1.-Algoritmo que lea números enteros hasta teclear 0, y nos muestre el máximo, el mínimo y la media de todos ellos. Piensa como debemos inicializar las variables.

```
Proceso ejercicio_17
 Escribir “ingrese numeros”
 Leer num
 minimo<-num
 maximo<-num
 suma<-0
 Mientras (num<>0) Hacer
  si (num>maximo) Entonces
  maximo<-num
 FinSi
 si (num<minimo) Entonces
  minimo<-num
 FinSi
 suma<-suma+num
 contador<-contador+1
 leer num
FinMientras
media<-suma/(contador)
```

```
 Escribir "El maximo es " maximo

 Escribir "El minimo es " minimo

 Escribir "La media es " media

FinProceso
```

2.-Dada una secuencia de números leídos por teclado, que acabe con un –1, por ejemplo: 5,3,0,2,4,4,0,0,2,3,6,0,……,-1; Realizar el algoritmo que calcule la media aritmética. Suponemos que el usuario no insertara numero negativos.

```
Proceso ejercici

 Escribir "ingrese"

 Leer num

 suma<-0

 contador<-1

 Mientras (num<>-1) Hacer

  suma<-suma+num

  contador<-contador+1

  Leer num

 FinMientras

 Escribir suma/(contador-1)

FinProceso
```

3.-Calcular independientemente la suma de los números pares e impares entre 1 y n

```
i<-1

sumapar=0

sumaimp=0

Escribir "ingrese un numero"
```

```
Leer n

Mientras i<=n Hacer

Si i mod 2=0 Entonces

sumapar<-sumapar+i

Sino

sumaimp<-sumaimp+i

Fin Si

i=i+1

Fin Mientras

Mostrar “la suma de los pares =”,sumapar

Mostrar “la suma de los impares =”,sumaimp

FinProceso
```

4.- Una tienda ofrece un descuento del 15% sobre el total de la compra durante el mes de octubre. Dado un mes y un importe, calcular cuál es la cantidad que se debe cobrar al cliente.

```
Proceso ejercicio

   Escribir “escribe el importe de la compra”

   Leer importe

   Escribir “Introduce el mes”

   Leer mes

   //Si el mes es octubre, se aplicara el descuento

   Si (mes=”octubre”) Entonces


 total<-importe*0.85

   Sino
```

total<-importe

FinSi

Escribir total

FinProceso

5.- Dados 10 números enteros que se ingresan por teclado, calcular cuántos de ellos son pares, cuánto suman ellos y el promedio de los impares

i=1

Mientras i<=10 Hacer

leer n

suma=suma+n

Si n mod 2=0 Entonces

sumapar<-sumapar+n

Sino

sumaimp<-sumaimp+n

Fin Si

i=i+1

Fin Mientras

Mostrar “la suma total es =”,suma

Mostrar “la suma de los pares =”,sumapar

Mostrar “la suma de los impares =”,sumaimp

FinProceso

6.- Crea una aplicación que nos pida un día de la semana y que nos diga si es un dia laboral o no. Usa un switch para ello.

Escribir “ingrese un dia de la semana”

Leer dia

Segun dia Hacer

“lunes”:

Mostrar “ES UN DIA LABORAL”

“martes”:

Mostrar “ES UN DIA LABORAL”

“miercoles”:

Mostrar “ES UN DIA LABORAL”

“jueves”:

Mostrar “ES UN DIA LABORAL”

“viernes”:

Mostrar “ES UN DIA LABORAL”

“sabado”:

Mostrar “no es un LABORAL”

“domingo”:

Mostrar “no es un LABORAL”

De Otro Modo:

Mostrar “escriba correctamente”

Fin Segun

FinProceso

7.- Muestra los números del 1 al 100 (ambos incluidos) divisibles entre 2 y 3. Utiliza el bucle que desees.

Para i<-1 Hasta 100 Hacer

Si i mod 3=0 y i mod 2 =0 Entonces

mostrar “numeros divisible para 3 y 2 es :”,i

Sino

si i mod 2 =0 Entonces

Mostrar “numeros divisible para 2 es :”,i

Sino

si i mod 3=0 Entonces

Mostrar “numero divisible para 3 es : “,i

FinSi

FinSi

Fin Si

Fin Para

FinProceso

8.-Cuenta la cantidad de vocales en forma independiente que se encuentren dentro de un texto introducido por el usuario

Escribir “ingrese un texto”

Leer text

Para i<-1 Hasta Longitud(text) Hacer

l<-Subcadena(text,i,i)

Si l=”a” Entonces

c=c+1

Sino

```
si l=”e” Entonces

c1=c1+1

Sino

si l=”i” Entonces

c2=c2+1

Sino

si l=”o” Entonces

c3=c3+1

Sino

si l=”u” Entonces

c4=c4+1

FinSi

FinSi

FinSi

FinSi

Fin Si

Fin Para

Mostrar “la cantidad de a =”,c

Mostrar “la cantidad de e =”,c1

Mostrar “la cantidad de i =”,c2

Mostrar “la cantidad de o =”,c3

Mostrar “la cantidad de u =”,c4

FinProceso
```

9.-Calcular las calificaciones de un grupo de alumnos. La nota final de cada alumno se calcula según el siguiente criterio: la parte práctica vale el 10%; la parte de problemas vale el 50% y la parte teórica el 40%. El algoritmo leerá el nombre del alumno, las tres notas, escribirá el resultado y volverá a pedir los datos del siguiente alumno hasta que el nombre sea una cadena vacía. Las notas deben estar entre 0 y 10, si no lo están, no imprimirá las notas, mostrara un mensaje de error y volverá a pedir otro alumno.

Repetir

Escribir “ingrese nombre del estudiante”

Leer nom1

Escribir “digite nota pactica (10%)”

Leer np

Si np<=1 Entonces

Escribir “digite nota problema (50%)”

Leer npr

si npr<=5 Entonces

Escribir “digite nota teorica(40%)”

Leer nt

si nt<=4 Entonces

re=np+npr+nt

Mostrar “nota final de “,nom1,” es igual a “,re

Sino

Mostrar “esta fuera de rango la nota teorica ”

FinSi

Sino

Mostrar “esta fuera de rango la nota problema ”

FinSi

Sino

Mostrar “esta fuera de rango la nota practica”

Fin Si

Hasta Que nom1=(“”)

FinProceso

10.-Calcula el Cuadrado y el Cubo de los 5 primeros números enteros que siguen a uno ingresado por teclado.

Proceso prog09

//9. Calcula el Cuadrado y el Cubo de los 5 primerosnúmeros enteros que siguen a uno ingresado por teclado

Escribir “ingrese un numero”

Leer nu

Para i<-nu Hasta nu+5 Hacer

doble=i*i

triple=i*i*i

Mostrar “el numero “,i,” al cuadrado es “,doble,” el cubo es = “,triple

Fin Para

FinProceso

Multiplica por 2 y divide entre 4 cualquier cantidad ingresada por teclado.

```
Proceso prog10
Escribir “ingrese el numero uno”
Leer n
res=n*2
div=res/4
Mostrar n,” multiplicado por 2 es igual a “, res
Mostrar “y dividido para 4 es igual a “,div
FinProceso
```

mostrar los números del 1 al 12 con los lazos para, repetir y mientras

```
Proceso prog08
Escribir “numeros 1 al 12 Con lazo mientras”
Mientras i<=11 Hacer
i=i+1
Mostrar i
Fin Mientras

Escribir “numeros 1 al 12 Con lazo repetir”
Repetir
a=a+1
Mostrar a

Hasta Que a>=11
```

```
Escribir “numeros 1 al 12 Con lazo para”

Para q<-1 Hasta 12 Hacer

Mostrar q

Fin Para

FinProceso
```

11.- Se desea obtener una operación básica matemática (+ , – , * , /) del 1 al 12, de cualquier número ingresado.

```
Proceso prog06

//6. Se desea obtener una operación básica matemática ( + , – , * , / ) del 1 al 12, de cualquier número ingresado

Escribir “(1)multiplicacion, (2)suma ,(3)resta, (4)division”

Escribir “escoja una opcion”

Leer num1

Segun num1 Hacer

1:

Escribir “multiplicacion”

Escribir “ingrese un numero”

Leer a

Para i<-1 Hasta 12 Hacer

Mostrar a,”*”,i,”=”,i*a

Fin Para

2:

Escribir “suma”

Escribir “ingrese un numero”

Leer q
```

Para i<-1 Hasta 12 Hacer

Mostrar q,"+",i,"=",i+q

Fin Para

3:

Escribir "resta"

Escribir "ingrese un numero"

Leer as

Para i<-1 Hasta 12 Hacer

Mostrar as,"-",i,"=",as-i

Fin Para

4:

Escribir "division"

Escribir "ingrese un numero"

Leer asd

Para i<-1 Hasta 12 Hacer

Mostrar asd,"/",i,"=",asd/i

Fin Para

De Otro Modo:

12.- Se desea validar una clave que sea 123456 hasta en tres oportunidades.

Proceso prog07

Escribir "ingrese la contraseña"

Repetir

i=i+1

Leer n

Hasta Que i=3 o n=(“123456”)

FinProceso

Ingrese Dos números para luego indicar si la suma es =24.

Proceso prog01

//ingrese dos números para luego indicar si

//la suma es =24?

n1=0

n2=0

s=0

Escribir “Ingrese primer numero”

Leer n1

Escribir “Ingrese segundo numero”

Leer n2

s=n1+n2

Si s=24 Entonces

Mostrar “la suma de “,n1,” y “,n2,” = “,”24”

Sino

Mostrar “no es igual a 24”

Fin Si

FinProceso

Determinar Si un Número es Múltiplo De 3.

Proceso prog02

13.- //Ingrese un numero y Determine si es múltiplo de 3

Escribir “Ingrese un numero”

Leer n

Si n mod 3=0 Entonces

Mostrar n,” es múltiplo de 3 ”

Sino

Mostrar n,” no es múltiplo de 3 ”

Fin Si

FinProceso

14.-Ingrese un número y Determine si s Par o Impar.

Proceso prog02

//Ingrese un número y Determine si s Par o Impar

Escribir “Ingrese un número”

Leer n

Si n mod 2=0 Entonces

Mostrar n,” es par ”

Sino

Mostrar n,” es impar ”

Fin Si

FinProceso

15.- Mostrar La tabla de multiplicar de cualquier número ingresado por teclado.

Proceso prog03

//Mostrar las tablas de multiplicar

// de cualquier numero ingresado por teclado

Escribir “Para Obtener las Tablas De Multiplicar”

Escribir “ingrese un numero”

Leer t

Para i<-1 Hasta 12 Hacer

Mostrar t,” * “,i,” = “,i*t

Fin Para

FinProceso

1.3.11. Sistemas formales

La teoría de autómatas y la teoría de funciones recursivas proveen modelos matemáticos que formalizan el concepto de algoritmo.
Los modelos más comunes son la máquina de Turing, máquina de registro y funciones u-recursivas. Estos modelos son tan precisos como un lenguaje de máquina, careciendo de expresiones coloquiales o ambigüedad, sin embargo se mantienen independientes de cualquier computadora y de cualquier implementación

1.3.12. Implementación

Muchos algoritmos son ideados para implementarse en un programa. Sin embargo, los algoritmos pueden ser implementados en otros medios, como una red neuronal, un circuito eléctrico o un aparato mecánico y eléctrico. Algunos algoritmos inclusive se diseñan especialmente para implementarse usando lápiz y papel. El algortitmo de multiplicación tradicional, elalgoritmo de Euclides, la criba de Erástones y muchas formas de resolver la raíz cuadrada son sólo algunos ejemplos.

1.3.13. Variables

Son elementos que toman valores específicos de un tipo de datos concreto. La declaración de una variable puede realizarse comenzando con var. Principalmente, existen dos maneras de otorgar valores iniciales a variables:
Mediante una sentencia de asignación.
Mediante un procedimiento de entrada de datos (por ejemplo: 'read').

Ejemplo:

```
 ...
i:=1;
read(n);
while i < n do begin
  (* cuerpo del bucle *)
  i := i + 1
end;
```

1.3.14. Estructuras secuenciales

La estructura secuencial es aquella en la que una acción sigue a otra en secuencia. Las operaciones se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. La asignación de esto consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:
Simples: Consiste en pasar un valor constante a una variable (a ← 15)
Contador: Consiste en usarla como un verificador del número de veces que se realiza un proceso (a ← a + 1)

Acumulador: Consiste en usarla como un sumador en un proceso (a ← a + b)

De trabajo: Donde puede recibir el resultado de una operación matemática que involucre muchas variables (a ← c + b*2/4).

Un ejemplo de estructura secuencial, Codificado en C++, como obtener la área de un triángulo:

```
Inicio
...
   float b, h, a;
   printf("Diga la base");
   scanf("%f", &b);
   printf("Diga la altura");
   scanf("%f", &h);
   a = (b*h)/2;
   printf("El área del triángulo es %f", a)
...
Fin
```

1.3.15. Estructuras Repetitivas

El computador está especialmente diseñado para aplicaciones en las que una operación o un conjunto de ellas deben repetirse muchas veces. En este sentido, definiremos bucle o lazo (loop), como un segmento de un programa cuyas instrucciones se repiten bien un número determinado de veces o mientras se cumpla una determinada condición.

Es imprescindible que se establezcan mecanismos para controlar esta tarea repetitiva, ya que si éstos no existen, el bucle puede convertirse en un proceso infinito. Así, en el bucle representado por el organigrama de la Figura 2, se observa que las instrucciones incluidas en él se repiten indefinidamente. El mecanismo de control citado se establece mediante una condición que se comprueba en cada paso o iteración del bucle.

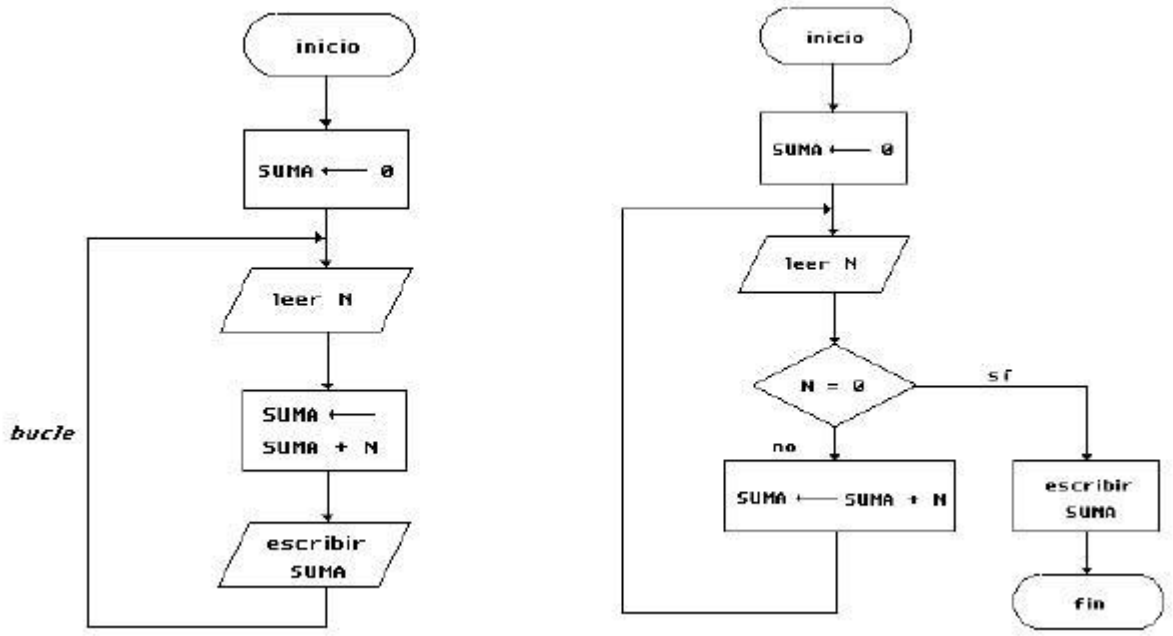


Figura 1

En la Figura 1 se coloca una condición tras la lectura de la variable N (comprobar si su valor es cero), de forma que tenemos la oportunidad de que el bucle deje de ser infinito, ya que podrá interrumpirse cuando la condición sea verdadera.

Los procesos que se repiten varias veces en un programa necesitan en muchas ocasiones contar el número de repeticiones habidas. Una forma de hacerlo es utilizar una variable llamada contador, cuyo valor se incrementa o decrementa en una cantidad constante en cada repetición que se produzca.

La Figura siguiente presenta un diagrama de flujo para un algoritmo en el que se desea repetir 50 veces un grupo de instrucciones, que llamaremos cuerpo del bucle, donde el contador se representa con la variable CONT. La instrucción que actualiza al contador es la asignación:

CONT <- CONT+1.
El contador puede ser positivo (incrementos de uno en uno) o negativo (decrementos de uno en uno).

En la Figura, el contador cuenta desde 1 a 50 y deja de repetirse cuando la variable CONT toma el valor 51 y termina el bucle.

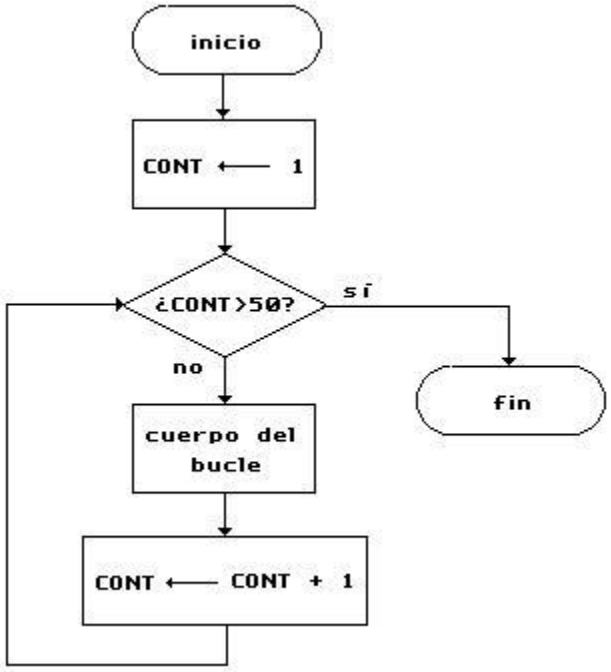


Figura 2

En la Figura siguiente se muestra un algoritmo que efectúa la operación de multiplicación n x m, sumando m un número n de veces. En él, el contador se decrementa: comienzaa contar en n y se va decrementando hasta llegar a cero; en ese momento se termina el bucle y se realiza la acción escribir.

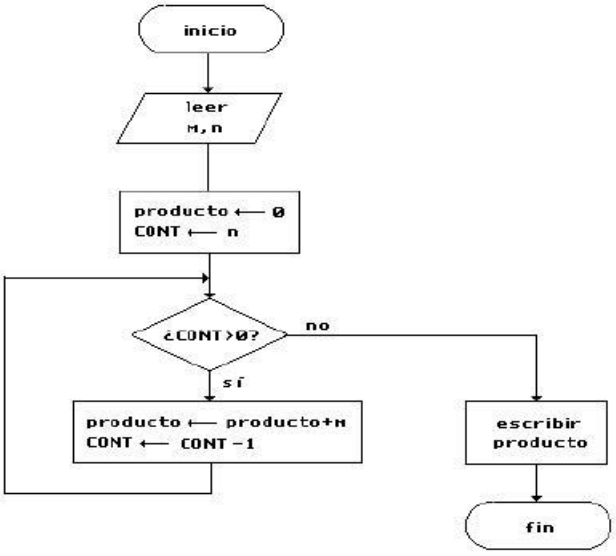


Figura 3

Otro tipo de variable, normalmente asociada al funcionamiento de un bucle es un acumulador o totalizador, cuya misión es almacenar una cantidad variable, resultante de operaciones sucesivas y repetidas. Un acumulador realiza una función parecida a la de un contador, con la diferencia de que el incremento o decremento, de cada operación es variable en lugar de constante.

Una estructura repetitiva es aquella que marca la reiteración de una serie de acciones basándose en un bucle. De acuerdo con lo anterior, esta estructura debe constar de tres partes básicas:

- decisión (para finalizar la repetición)
- cuerpo del bucle (conjunto de instrucciones que se repiten)
- salida del bucle (instrucción a la que se accede una vez se decide finalizar)

Tomando el caso anterior, donde para obtener la suma de una serie de números, hemos utilizado la estrategia siguiente: tras leer cada número lo añadimos a una variable SUMA que contenga las sucesivas sumas parciales (SUMA se hace igual a cero al inicio). Observemos que el algoritmo correspondiente deberá utilizar sucesivamente instrucciones tales como:

```
leer número
si N < 0 entonces
  escribir SUMA
  si-no
SUMA  <- SUMA+número
fin-si
```

que se pueden repetir muchas veces; éstas constituyen el cuerpo del bucle.

Una vez se ha decidido el cuerpo del bucle, se plantea la cuestión de cuántas veces se debe repetir. De hecho conocemos ya la necesidad de contar con una condición para detener el bucle. En el Ejemplo 8, se pide al usuario el número N de números que desea sumar, esto es, el número de iteraciones del bucle.
Usamos un contador de iteraciones, TOTAL, que se inicializa a N y a continuación se decrementa en uno cada vez que el bucle se repite; para ello introducimos una acción más al cuerpo del bucle: TOTAL <- TOTAL -1. También podríamos inicializar la variable TOTAL en 0 o en 1, e ir incrementándolo en uno, en cada iteración, hasta llegar al número deseado N.

Ejemplo:

Hallar la suma de N números, a través de una estructura repetitiva

```
algoritmo suma_números
{leer número total de números a sumar en variable N}
TOTAL <-   N
SUMA <-  0   { la suma parcial es 0 al inicio}
{comienzo de bucle}
```

```
mientras que TOTAL > 0 hacer
leer número
SUMA <-  SUMA+número
TOTAL <-  TOTAL-1
fin_mientras
{fin del bucle}
escribir "la suma de los" , N , "números es " , SUMA
```

Aunque la condición de finalización puede evaluarse en distintos lugares del algoritmo, no es recomendable que ésta se pueda efectuar a mitad del cuerpo del bucle, por lo que es bueno que se produzca al principio o al final del mismo. Según donde se sitúe la condición de salida, dará lugar a distintos tipos de estructuras repetitivas que analizaremos a continuación: estructura desde-hasta, estructura mientras y estructura repetir-hasta_que.

1.3.16. ESTRUCTURA DESDE-HASTA

Esta estructura consiste en que la condición de salida se basa en un contador que cuenta el número de iteraciones. Por ejemplo, el ejemplo 8 podría hacerse de la siguiente manera:

```
desde i = 1 hasta N con_incremento 1 hacer
        leer número
        SUMA <-  SUMA + número
fin_desde
```

donde i es un contador que cuenta desde un valor inicial (1) hasta el valor final (N) con los incrementos que se consideren (de uno en uno en este caso). Esta es la llamada estructura Desde ("for"), que es la más simple desde el punto de vista de la condición de salida, ya que viene predeterminada por el código. Su utilidad reside en el hecho de que, en muchas ocasiones, se conoce de antemano el número de iteraciones. Esta estructura ejecuta las acciones del cuerpo del bucle, un número especificado de veces y de modo automático controla el número de iteraciones. Su formato en pseudocódigo es:

```
desde v=vi hasta vf hacer
  <acciones>
  .
  .
fin_desde
```

v: variable índice
vi, vf: valores inicial y final de la variable

La variable índice o de control normalmente será de tipo entero y es normal emplear como identificador, las letras I,J,K como herencia de los índices y subíndices utilizados en cálculo científico. El incremento de la variable índice es 1 en cada iteración si no se indica expresamente lo contrario. Si debemos expresar incrementos distintos de +1 el formato de la estructura es:

```
desde v = vi hasta vf inc incremento hacer
  <acciones>  si vi > vf entonces usar
```

. en lugar de inc incremento
. la expresión dec decremento
fin_desde

Obviamente, si el valor inicial de la variable índice es menor que el valor final, los incrementos deben ser positivos, ya que en caso contrario la secuencia de acciones no se ejecutaría. De igual modo si el valor inicial es mayor que el valor final, el incremento debe ser en este caso negativo. Así:

desde I=20 hasta 10 hacer
<acciones>
fin_desde

for(v=vi; v<=vf; vt+=step)
<accion o bloque de acciones >
(más genéricamente)
for(inicio;condición;actualización)
<accion o bloque de acciones >

1.3.17. ESTRUCTURA MIENTRAS

Si la condición es verdadera, entonces se ejecuta el cuerpo del bucle. No todos los lenguajes incluyen
la estructura mientras.
En lenguaje C:
Cuando la condición de salida del bucle se realiza al principio del mismo, éste se ejecuta mientras se verifica una cierta condición. Es la llamada estructura repetitiva mientras ("while"); en ella el cuerpo del bucle se repite mientras se cumple una determinada condición. Su pseudocódigo es:

mientras condición hacer
<acciones>
fin_mientras

Cuando se ejecuta la instrucción mientras, la primera cosa que sucede es la evaluación de la condición. Si es falsa, no se ejecuta ninguna acción y el programa prosigue en la siguiente instrucción a la finalización del bucle; si la condición es

while (condicion)
<accion>

Obsérvese que en una estructura mientras si la primera evaluación de la condición es falsa, el cuerpo del bucle nunca se ejecuta. Puede parecer inútil ejecutar el cuerpo del bucle cero veces, ya que no tendrá efecto en ningún valor o salida; sin embargo, puede ser una acción deseada. Por ejemplo el siguiente bucle para procesar las notas de unos examenes contando el número de alumnos presentados dejará de ejecutarse cuando el numero leído sea negativo. Si la primera nota introducida fuera negativa, la acción deseada es, efectivamente, que no se ejecute el bucle ninguna vez.

```
C  <- 0
leer nota
mientras nota = 0 hacer
{procesar nota}
C  <- C+1
leer nota
fin mientras
```

1.3.18. ESTRUCTURA REPETIR-HASTA_QUE

En esta estructura la condición de salida se sitúa al final del bucle; el bucle se ejecuta hasta que se verifique una cierta condición. Es la llamada estructura Repetir-hasta ("repeat-until"). Existen muchas situaciones en las que se desea que un bucle se ejecute al menos una vez, antes de comprobar la condición de repetición. Para ello la estructura repetir-hasta_que se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle. En pseudocódigo se escribe:

```
repetir
  <acciones>
hasta_que <condición>
```

```
do
<accion o bloque>
while (condicion);
```

(en C se "dice" mientras se cumpla la condición, no hasta que se cumpla)

El bucle repetir-hasta_que se repite mientras la condición sea falsa, justo lo opuesto a la estructura mientras. Sea el siguiente algoritmo:

```
inicio
        contador <- 1
        repetir
        leer número
        contador  <- contador+1
        hasta_que contador > 30
        escribir "números leídos: 30"
fin
```

Este bucle se repite hasta que el valor de variable contador exceda a 30, lo que sucederá después de 30 ejecuciones del mismo. Nótese que si en vez de 30 pusiéramos 0, el cuerpo del bucle se ejecutará siempre al menos una vez.

Concepto Clave:

- Las estructuras de carácter repetitivo permiten realizar la ejecución de grupos de instrucciones varias veces.
- La diferencia principal entre la sentencia **repetir hasta** y **mientras** es que la **repetir hasta** se realizará al menos una vez , en tanto la **mientras** se realizara cero ò mas veces

Ejemplo:

Calcular el factorial de un número N, usando la estructura repetir.

```
inicio
        leer N
        Factorial  <-1
        I <-  1
        repetir
                Factorial  Factorial * I
                I <-  I+1
        hasta_que I > N
        escribir "el factorial del número", N, "es", Factorial
fin
```

A)

```
v <-  vi
mientras v < = vf hacer
        <acciones>
  v <-  v + incremento
  fin_mientras
```

B)

```
v <-  vi
mientras v > = vf hacer
                <acciones>
  v <-  v - decremento
        fin_mientras
```

Concepto Clave:

Las tres estructuras repetitivas son susceptibles de intercambio entre ellas, así por ejemplo es posible, sustituir una estructura desde, por una mientras; con incrementos positivos o negativos de la variable índice. En efecto, la estructura desde con incremento positivo es equivalente a la estructura mientras marcada con la A), y la estructura desde con incremento negativo es equivalente a la estructura mientras marcada con la B).

Ejemplo

Calcular los factoriales de n números leídos por el teclado.

El problema consiste en realizar una primera estructura repetitiva de n iteraciones del algoritmo de cálculo del factorial, que a su vez se efectúa con una segunda estructura repetitiva.

```
inicio
leer n  {lectura de la cantidad de números}
desde i = 1 hasta n hacer
leer NUMERO
```

```
FACTORIAL <- 1
desde j = 1 hasta NUMERO hacer
FACTORIAL<-  FACTORIAL *j
fin_desde
escribir “el factorial del número”, NUMERO, “es”, FACTORIAL
fin_desde
fin
```

1.3.19. Ejercicios resueltos pseudocodigo

1.- Escribir un Pseudocódigo de un programa que permita leer la edad y peso de una persona y posteriormente imprimirla.

Inicio

Variables edad, peso.

Imprimir "Escribir los datos (Edad, Peso):"

Leer Edad, Leer Peso.

Visualizar "Tu peso es: ", peso, " y tu edad es: ", edad.

Fin.

2.- Escribir un Pseudocódigo que calcule el área de un triángulo recibiendo como entrada el valor de base y altura.

Inicio

Variables Altura, Base, Area.

Imprimir "Introduce la base y la altura: "

Leer base y altura.

Area= (base*altura)/2.

Imprimir "El area es: ", Area

Fin.

3.- Escribir Pseudocódigo que calcule el área de un círculo.

Inicio

Constantes Pi= 3.1416

Variables Radio, area = real

Imprime "Introduce el radio: "

Leer radio.

area= radio*radio*Pi

Imprimir "El área del circulo es: ", area.

Fin.

4.- Escribir Pseudocodigo que dados 2 valores de entrada imprima siempre la división del mayor entre el menor.

Inicio

Variables num1, num2=entero.

Variables R=real.

Imprimir "Introduce los números:"

Leer num1, Leer num2.

Si a>b entonces

R= a/b

Sino

R= b/a

Finsi

Imprimir "La división es =", R;

5.- Escribir Pseudocódigo que lea de entrada 3 números y que indique cual es el mayor de ellos.

Inicio

Variables a, b,c = enteros.

Imprimir "Introduce los datos a comparar: "

Leer a,b,c

Si a<b y a<c entonces

Imprime "El mayor es: ", a

Sino

Si b>a y b>c entonces

Imprime "El mayor es: ",b

sino

Imprime "El mayor es: ", c

Fin.

6.- Escribir un Pseudocódigo que lea 3 números los cuales significan una fecha (día, mes, año). Comprobar que sea válida la fecha, si no es válido que imprima un mensaje de error, y si es válida imprimir el mes con su nombre.

Inicio

Variables dia, mes, año =entero.

Imprimir "Introduce la fecha (Dia,mes,año): "

Leer dia

leer mes

leer año.

Si dia>31 o mes >12 o año<0 entonces

Imprimir "Error la fecha no es correcta"

Sino

si mes=1 o mes=3 o mes=5 o mes=7 o mes=8 o mes=10 o mes=12 entonces

si dia >31 y dia<1 entonces

Imprimir "Error de Dia"

```
sino
si mes=1 entonces imprimir dia,"/ Enero /",año
si mes=3 entonces imprimir dia,"/ Marzo /",año
si mes=5 entonces imprimir dia,"/ Mayo /",año
si mes=7 entonces imprimir dia,"/ Julio /",año
si mes=8 entonces imprimir dia,"/ Agosto /",año
si mes=10 entonces imprimir dia,"/ Octubre /",año
si mes=12 entonces imprimir dia,"/ Diciembre /",año
Si mes=2 entonces
si dia>28 o dia<0 entonces
Imprimir "Error de Dia"
sino
Imprimir dia,"/ Febrero /",año
 Si mes=4 o mes=6 o mes=9 o mes=11
Si dia >30 o dia <1 entonces
Imprimir "Error de Dia"
Sino
Si mes=4 entonces
Imprimir dia,"/ Abril /",año
Si mes=6 entonces
Imprimir dia,"/ Junio /",año
Si mes=9 entonces
```

Imprimir dia,"/ Septiembre /",año

Si mes=11 entonces

Imprimir dia,"/ Noviembre /",año

Fin.

7.- Escribir un Pseudocodigo que pida la edad y el sexo y dependiendo si es hombre o mujer y si puede votar o no.

Inicio

variables edad=entero, sexo=caracter.

repetir

Imprimir "Introduce Tu edad:"

Leer edad.

Hasta que (edad >0)

Hacer

Imprimir "Introduce tu sexo (M/H):"

leer sexo

hasta que (sexo='H' o sexo = 'M')

Si sexo= 'M' entonces

si edad > 18 entonces

Imprimir "Eres Mujer y puedes votar"

sino

Imprimir "Eres Mujer y no puedes votar"

Sino

si edad >18 entonces

Imprimir "Eres Hombre y puedes votar"

sino

Imprimir "Eres Hombre y no puedes votar"

Fin.

8.- Realice un Pseudocódigo que calcule la nómina salarial neto, de unos obreros cuyo trabajo se paga en horas. El cálculo se realiza de la siguiente forma:

- Las primeras 35 horas a una tarifa fija.

- Las horas extras se pagan a 1.5 más de la tarifa fija.

- Los impuestos a deducir de los trabajadores varian, segun el sueldo mensual si el sueldo es menos a $20,000.00 el sueldo es libre de impuesto y si es al contrario se cobrará un 20% de impuesto.

Inicio

Constante Tarifa= 50.

Variables Horas, Sueldo, dif_horas, tarifa_extra, Salario_extra, Sueldo_mes, Impuesto, Sueldo_total.

Imprimir "Introduce las Horas de Trabajo ==> "

Leer Horas.

Si Horas <= 35 entonces

Sueldo= horas * tarifa.

Sino

Dif_horas= Horas - 35

tarifa_extra=(tarifa * 1.5)

Salario_exta= tarifa_extra * Dif_horas

Sueldo=(35*tarifa)+ salario_extra.

Sueldo_mes = sueldo * 4

Impuesto=0

Si sueldo_mes > 20000 entonces

Impuesto = (Sueldo_mes* 0.20)

Salario_total = Sueldo_mes - Impuesto

Imprimir "Tu sueldo al mes es: ", sueldo_mes,"Valor de Impuesto: ", Impuesto, " El salario Neto es: ",Salario_total.

sino

Imprimir "Tu sueldo Neto es: ", sueldo_mes.

Fin.

11.- Escribir un Pseudocódigo que encuentre y despliegue los números primos entre uno y cien. Un número primo es divisible entre el mismo y la unidad por lo tanto un numero primo no puede ser par excepto el dos (2).

Inicio

Variables primo, cont, div, res.

Primo = 1.

Hacer mientras primo<=100

Div =0.

Cont =1.

Hacer Mientras cont <= primo

Res = cont mod primo

si res = 0 entonces

Div = div +1.

Fin si

Cont = cont +1.

Fin de Hacer mientras

si div<=2 entonces

imprimir primo

Fin si.

Primo = primo + 1.

Fin de Hacer mientras

Fin.

9.- Hacer un Pseudocódigo que calcule el factorial de un número.

Inicio

Variables N,Fact,aux.

Imprimir "Introduce el número: "

Leer N

aux= n-1.

fact=n

Hacer

fact=fact * aux.

Hasta que aux=1

Imprimir "El factorial de ", n, "es:", fact

Fin.

10.- Hacer un Pseudocodigo que despliegue las tablas de multiplicar.

Inicio

```
Variables i,k,r.

para i=1 hasta 10.

para k=1 hasta 10.

r:=i*k.

Imprimir i," por ",k," = ",r.

k=k+1.

i=i+1.

Fin.
```

Capítulo No 2:

2. Programas y Programación

2.1. PROGRAMAR LAS TAREAS COTIDIANAS:

Quieres que el televisor te despierte a las 3 am? Entonces prográmalo para que se encienda automáticamente.
¿Vas a salir de viaje y no tendrás acceso a tu email, entonces programa tu email para decir que no estas disponible?

Figura 4

"En general podríamos decir que un programa se asemeja a una especie de máquina que al ponerse en marcha sobre un determinado conjunto de datos, permite solucionar un determinado problema."
Esto cómo funciona?: Simplemente escojo la opción adecuada de mi control remoto
Esta máquina se construye a partir de un conjunto de piezas básicas (sentencias), cada una de las cuales tiene una funcionalidad dentro de la misma (semántica).
Si pasamos de la similitud a la definición podríamos decir que Programa es: "Un conjunto de sentencias, que ejecutadas sobre un determinado conjunto de datos, produce los resultados deseados con respecto a una especificación de un problema" (Quereda, 2000)
Por lo tanto la acción de programar significa determinar una secuencia y ordenar una serie de sentencias para resolver un problema como lo ilustra el grafico:

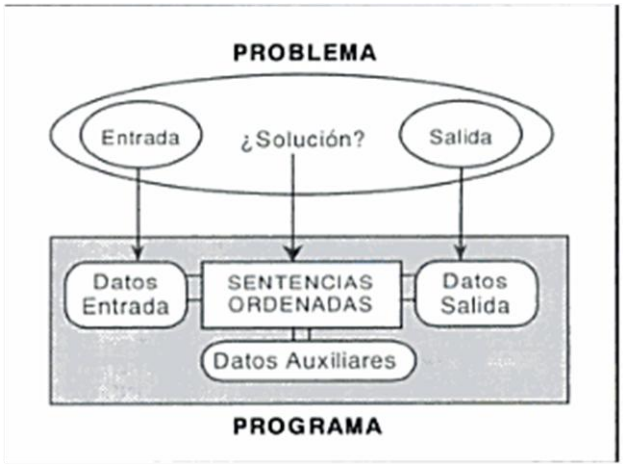


Figura 5

2.2. Lenguaje de Programación

2.1.1. Tipos de Lenguajes

Lenguaje C

El Lenguaje C fue diseñado por Dennies Ritchie en 1970, en los laboratorios Bell de Estados Unidos.

Este lenguaje presenta varias características, entre las cuales están:

1. Lenguaje de programación de propósitos generales
2. Permite la Programación Estructurada
3. Abundancia de Operadores y Tipos de Datos
4. No está asociado a ningún sistema operativo ni a ninguna máquina
5. Popular y Eficaz
6. Permite el desarrollo de Sistemas Operativos y programas de aplicación
7. Portabilidad
8. Existen las librerías en las bibliotecas
9. tiene sólo 32 palabras reservadas

2.1.2. CODIGO FUENTE:

[7]Programa fuente o código fuente: es una noción que se emplea como sinónimo de código fuente. Se trata de las instrucciones que un programa informático transmite a una computadora para que pueda ejecutarse. Dichas instrucciones son líneas de texto escritas en un lenguaje de programación (una estructura capaz de impartir instrucciones informáticas a partir de una determinada base semántica y sintáctica).[8]

Figura 6

2.1.3. Compiladores

[9]

Un compilador es un programa informático, que se encarga de traducir el código fuente de una aplicación que este en desarrollo, es decir convierte un programa hecho en lenguaje de programación de alto nivel a un lenguaje de máquina, el cual es conocido como de bajo nivel, de tal forma que sea más entendible y mucho más fácil de procesar en el equipo en el que se está ejecutando.

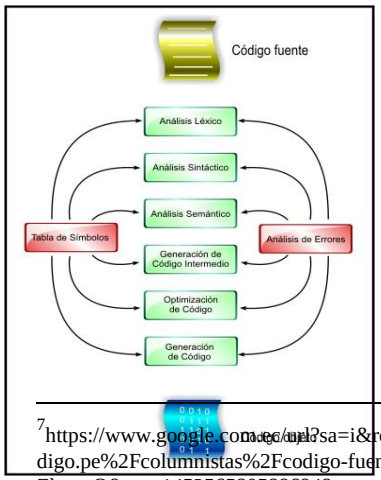


Figura 7

[7]https://www.google.com.ec/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=&url=http%3A%2F%2Fwww.codigo.pe%2Fcolumnistas%2Fcodigo-fuente%2Fel-codigo-fuente-de-la-publicidad%2F&psig=AFQjCNHAoz1lzp9Bqk1OL-QwFrj-Zh_unQ&ust=1452565805886948

[8]http://definicion.de/programa-fuente/

[9]http://2.bp.blogspot.com/-IDAgRNEuofg/Tm7TQysV9EI/AAAAAAAAABU/uEBNoEMbW_M/s1600/compilador.jpg

2.1.4. Traductores

De igual manera un traductor es el que toma como entrada un texto escrito y da como salida otro texto en un lenguaje llamado objeto.

2.1.5. CODIGO OBJETO Y EJECUTABLE

Código objeto: Conjunto de instrucciones y datos escritos en un lenguaje que entiende el ordenador directamente: binario o código máquina. Provienen de la traducción de cierto código fuente, es un fragmento del programa final y es específico de la plataforma de ejecución.

Código ejecutable: Reúne diferentes códigos u objetos generados por los programadores junto con las "librerías de uso general" (propias del entorno o del lenguaje de programación) componiendo el programa final. Este es el código que ejecutan los usuarios del sistema, y es específico para una plataforma concreta: Windows, Linux, Mac OS, o cierto sistema Hardware.[10]

Figura 8[11]

Capítulo No 3: Lenguaje C

3.1. LENGUAJE C

El Lenguaje C fue diseñado por Dennies Ritchie en 1970, en los laboratorios Bell de Estados Unidos.

Este lenguaje presenta varias características, entre las cuales están:
1.Lenguaje de programación de propósitos generales
2.Permite la Programación Estructurada
3.Abundancia de Operadores y Tipos de Datos
4.No está asociado a ningún sistema operativo ni a ninguna máquina
5.Popular y Eficaz

[10] http://entornosdedesarrollo.blogspot.com/2011/11/codigo-fuente-codigo-objeto-y-codigo.html

[11] https://www.google.com.ec/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjmlZq_9KHKAhXH1R4KHcM6A5sQjRwIBw&url=https%3A%2F%2Fes.wikipedia.org%2Fwiki%2FC%25C3%25B3digo_objeto&psig=AFQjCNFaVtvVCMHLYuSDPOqA0BFiL5DZJg&ust=1452606888429380

6.Permite el desarrollo de Sistemas Operativos y programas de aplicación
7.Portabilidad
8.Existen las librerías en las bibliotecas
9. Tiene sólo 32 palabras reservadas

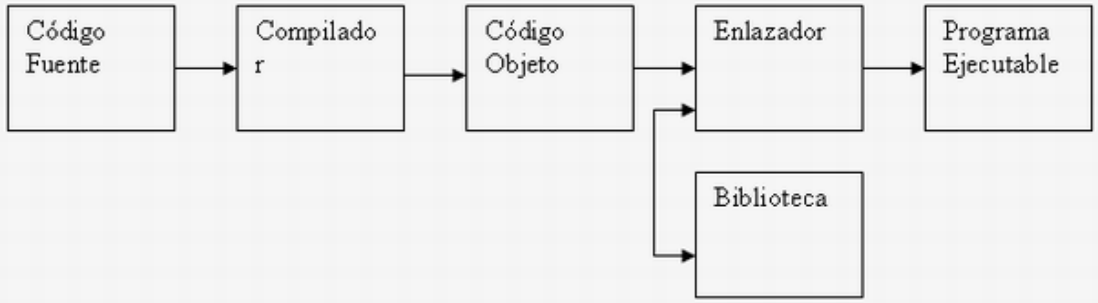


Figura 9[12]

Bibliotecas: es el archivo que contiene código objeto de una colección de rutinas o funciones que realizan tareas determinadas y se pueden utilizar en los programas.

Enlazador: Programa que convierte las funciones de la biblioteca estándar de C, con el código que ha traducido el compilador.

3.2. ESTRUCTURA DE UN PROGRAMA EN C

Directivas de pre-procesador (instrucciones que se le dan al compilador #include antes de compilar)

#define

ejemplo:

#include <stdio.h>

Lo que se le esta indicando, es que de las librerías, "Incluya" en nuestro programa la directiva stdio.h, la cual contiene las funciones de entrada y salida de datos (standar input output, en inglés). Si necesitamos las funciones matemáticas, debemos especificarlo con la declaratoria:

#include <math.h>

Si necesitamos las funciones de cadenas:

#inlcude <stlib.h>

[12] http://www.monografias.com/trabajos33/programacion-lenguaje-c/Image5733.gif

Es necesario aclarar que esto se hace al inicio del programa, y las declaratorias deben llevar el símbolo de numeral (#) seguido de la sentencia "include", y entre signosde mayor y menor que (<>) el nombre de la directiva.

3.3. Declaraciones Globales:

*Prototipos de Funciones: También llamadas declaraciones de funciones, lo cual se tratará más adelante

.1. **Declaraciones de Variables:** cabe destacar, que esto se hace seguido de los #include y los #define.

Función Principal main(): Esta es la función principal de nuestro programa, su cuerpo, por ello NUNCA debe faltar, ya que en ella van contenidas todas las instrucciones de nuestro programa.

```
main()

{

declaraciones locales /*Comentarios */

sentencias

}
```

la función main() va al inicio, luego abrimos llaves y dentro de ellas van las declaraciones de variables, las sentencias de lectura , cálculos, asignaciones e impresiones, y con la última llave (}), le indicamos el final del programa.

Ejemplo:

Programa que a partir del radio, calcula el área de un circulo

```
#include <stdio.h>

#include <conio.h>

main()

{

float radio, area;

printf("Radio=\n");
```

```
scanf("%f", &radio);

area=3.14159*radio*radio;

printf("El Area es %f\n\n", area);

getch();

return 0;

}
```

Explicación:

Le indicamos al compilador, que usaremos las bibliotecas <stdio.h> y <conio.h>, ¿por qué <conio.h>?, por que esta biblioteca, contiene las funciones getche(), getch(), etc, y de una de ellas hacemos uso en este pequeño ejemplo.

Luego, le indicamos a nuestro programa el incio de nuestro programa (función main()).

Declaramos, como valores reales, las variables radio y area (de esto se hablará más adelante). Luego, con la instrucción printf(), mostramos en pantalla el mensaje (Radio=) y scanf se encarga de leer el valor digitado por el usuario. Posteriormente area, es igual al la multiplicación de pi (3.14159), el radio al cuadrado. Se muestra en pantalla ese resultado, luego el programa espera que se presiones cualquier tecla (getch()) y no retorna ningún valor (return 0).

3.4. SINTAXIS- Elementos de Un Programa en C

Como su nombre lo indica, estos son los nombres, con los que identificamos las variables, constantes, funciones, vectores, etc, de nuestro programa. Para ello debemos tener presente algunas reglas:

- pueden tener de 1 hasta un máximo de 31 caracteres
- Debe de iniciar con una letra o subrayado

Ejemplo:

(Correctos)

c2
_c2

(Incorrectos)

2c

2 c

- No es lo mismo una minúscula que una mayúscula, ya que c distingue de entre ellas. Ejemplo: BETA ¹ Beta ¹ beta ¹ BeTa

- No son válidos los identificadores de palabras reservadas. En un inicio hablamos que c posee 32 palabras reservadas, entre ellas están:

float char while

int else return

Estas palabras no pueden ser utilizadas para identificar variables, constantes, funciones etc

3.5. Identificadores:

En todo programa que estemos diseñando en C (o en cualquier otro lenguaje de programación); es necesario insertar ciertos comentarios en el código, para que en posteriores modificaciones y cuando se realice el mantenimiento, podamos recordar cosas importantes ya que, en los comentarios, podemos incluir aspectos importantes del programas, explicaciones del funcionamiento de las sentencias, etc.

El formato de los comentarios en C, es el siguiente:

/*este es un comentario en C */

/*Podemos colocar mucha información importante

de nuestro Programa */

Comentarios

Permite que, el pre-procesador, incluya funciones proporcionadas por el fabricante, a nuestro programa. Ejemplo:

#include <stdio.h> /* le decimos al compilador que incluya la librería

stdio.h */

3.6. Nombre de indentificadores

Son los nombres usados para referirse a las variables, funciones, etiquetas y otros objetos definidos por el usuario.

La longitud de un identificador en Turbo C puede variar entre 1 y 32 caráteres. El primer carácter debe ser una letra o un símbolo de subrayado, los carácteres siguientes pueden ser letras, números o símbolos de subrayado.

Correcto -----> cont, cuenta23, balance_total
Incorrecto -----> 1cont, hola!, balance...total

3.7. La Directiva #include

permite definir constantes simbólicas. Pero hasta ahora ha sido poco lo que hemos hablado acerca de las constantes, es por ello que aprovechando, este espacio; dedicaré unas cuantas líneas para aclarar ello.

Las variables pueden cambiar de valor, durante la ejecución del programa, por eso es que se llaman variables. Y las constantes como su nombre lo indica, son valores que permanecen constantes durante toda la ejecución del programa, un ejemplo de ello, es el valor de (pi) que equivale a 3.14159....

En C existen diferentes tipos de variables, entre ellas tenemos:

3.8. Constates Numéricas:

Son valores numéricos, enteros o de reales (de punto flotante). Se permiten también constantes octales y hexadecimales.

Concepto Clave:
En C las mayúsculas y las minúsculas se tratan como distintas

3.9. Constantes Simbólicas:

Las constantes simbólicas tienen un nombre (identificador), y en esto se parecen las variables. Sin embargo, no pueden cambiar de valor a lo largo de la ejecución del programa. En C, se pueden definir mediante el preprocesador.

(Tomado del Manual "Aprenda Lenguaje ANSI C como si estuviera en Primero" Escuela superior de Ingenieros Industriales. Universidad de Navarra. Febrero de 1998).

Ejemplo:

```
#define N 100
```

#define PI 3.1416

1.#define B 45

Esta directiva (#define) va, inmediatamente después de los #include. Se escribe la directiva, se deja un espacio y se escribe el identificador de la constante, otro espacio y su valor.

2. la directiva #define
/ ! % ^ & * () - + {} [] \ ; : <> ¿ .

3. Signos de Puntuación y de Separación
Regla de Oro: Al momento de programar en C, esta es una regla de oro, y la causa por la cual nuestro programa puede darnos muchos errores de sintaxis, cuando se omite, al final de cada sentencia un punto y coma (;). Ya que con ello le indicamos al compilador que ha finalizado una sentencia.

NOTA: el lector no debe confundirse, las directivas: #include, #define. Main(), no llevan punto y coma, porque no son sentencias.

Concepto Clave:
Regla de Oro: Se debe considerar el uso de (;) al final de cada instrucción en C.
La causa por la cual nuestro programa puede darnos muchos errores de sintaxis, cuando se omite, al final de cada sentencia un punto y coma (;). Ya que con ello le indicamos al compilador que ha finalizado una sentencia.

Recordemos el ejemplo 1.1, y vea que al final de cada sentencia lleva su correspondiente punto y coma:

```
#include <stdio.h>

#include <conio.h>

main()

{

float radio, area;

printf("Radio=\n");
```

```
scanf("%f", &radio);

area=3.14159*radio*radio;

printf("El Area es %f\n\n", area);

getch();

return 0;

}
```

4. Todas las Instrucciones o sentencias del programa terminan con un punto y coma (;) Esta consideración toma mayor auge, cuando veamos las instrucciones anidadas en condiciones, ciclos, etc.

Ejemplo:

```
{
...

printf("Hola\n\b");

...
}
```

5.Todo Bloque de Instrucciones debe ir entre llaves
6.En una línea se pueden escribir más de una instrucción separada por un punto y coma

7.Esto es posible, porque con el punto y coma, le estamos indicando al compilador el fin de una sentencia o instrucción.

Ejemplo:

```
b = c + d; d = 2*k;
```

3.10. Tipos de Datos en C

Un tipo de dato, se define como *un conjunto de valores que puede tener una variable, junto con ciertas operaciones que se pueden realizar con ellas.*

Tipos básicos de datos:

Tipo	Descripcion	Modificadores
void	Vacio	
char	Caracter (8 bits)	signed char(8 bits), unsigned char(8 bits)
int	Entero simple (16 bits)	signed int(16 bits), unsigned int(16 bits), long int (32 bits), unsigned long int(32 bits), signed long int(32 bits), short int(16 bits), unsigned short int(16 bits), signed short int(16 bit)
float	Coma flotante (32 bits)	
double	Coma flotante mas grande (64 bits)	long double (80 bits)
bool	Valor booleano: true o false	
wchar_t	Caracteres anchos, para determinado juegos de caracteres	

[13]

[13](Class, 2004)

*TIPOS DE DATOS PREDEFINIDOS

TABLA CON LOS TIPOS DE DATOS PREDEFINIDOS EN C			
ENTEROS: números completos y sus negativos			
Palabra reservada:	**Ejemplo**	**Tamaño (byte)**	**Rango de valores**
int	-850	2	-32767 a 32767
VARIANTES DE ENTEROS			
short int	-10	1	-128 a 127
unsigned int	45689	2	0 a 65535
long int	588458	4	-2147483648 a 2147483647
unsigned long	20000	4	0 a 4294967295
>REALES: números con decimales o punto flotante			
Palabra reservada:	**Ejemplo**	**Tamaño (byte)**	**Rango de valores**
float	85	4	$3.4x10^{-38}$ a $3.4x10^{38}$
VARIANTES DE LOS REALES			
double	0.0058	8	$1.7x10^{-308}$ a $1.7x10^{308}$
long double	1.00E-07	10	$3.4x10^{-4932}$ a $1.1x10^{4932}$
>CARÁCTER: letras, digitos, símbolos, signos de puntuación.			
Palabra reservada:	**Ejemplo**	**Tamaño (byte)**	**Rango de valores**
char	'O'	1	0 255

NOTA: El tipo de dato string y bolean NO existen en C.

3.11. Declaración de Variables

Una Variable, como su nombre lo indica, es capaz de almacenar diferentes valores durante la ejecución del programa, su valor varía. Es un lugar en la memoria el cual, posee un nombre (identificador), y un valor asociado.

La declaración de variables en C, se hace en minúsculas.

Formato:

Tipo_de_dato nombre_de_la_variable;

Ejemplos:

*Declare una variable de tipo entero y otra de tipo real, una con el nombre de "x" y otra con el identificador "y":

int x;

float y;

*Declare una variable de tipo entero llamada moon, e inicialícela con un valor de 20

int moon = 20;

*Declare una variable de tipo real, llamada Pi, e inicialícela con una valor de 3.1415

float pi=3.1415;

*Declare una variable de tipo caracrter y asígnele el valor de "M"

char car = 'M';

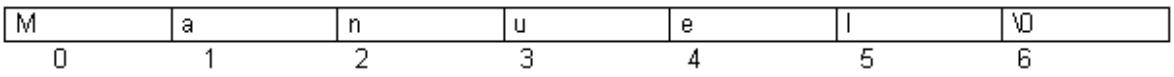
*Declare una variable llamada nombre, que contenga su nombre:

char nombre[7]="Manuel";

Explicación:

En el apartado anterior, se explicó, que C, no tiene el tipo de dato llamado string, o mejor conocido como cadenas de texto, pero nosotros podemos hacer uso de ellas, por medio de un arreglo, (de lo cual hablaremos con más detalle, posteriormente); pero para declarar este tipo de datos colocamos el tipo de datos, es decir la palabra reservada char luego el nombre, e inmediatamente abrimos, entre corchetes, va el número de letras, que contendrá dicha variable. Es muy importante que al momento de declarar el tamaño, sea un número mayor, al verdadero número de letras; por ejemplo, la palabra "Manuel", solo tiene 6 letras, pero debemos declararlo para 7 letras ¿Por qué?.

Veámoslo gráficamente, en la memoria, C crea un variable llamada nombre y esta posee la palabra Manuel, así:



en realidad, hay 7 espacios, pero la cuenta llega hasta 6, por que c, toma la primera posición como la posición cero, y para indicar el final de la cadena lo hace con un espacio en blanco.

El \0 indica el fin de datos.

3.12. Declaración de Constantes

Las constantes, como su nombre lo indica, son valores que se mantiene invariables durante la ejecución del programa.

Su formato es el siguiente:

const tipo_de_dato nombre= valor;

donde const, es una palabra reservada, para indicarle al compilador que se esta declarando una constante.

Ejemplo:

const int dia=7;

const tipo_de_dato nombre= valor;

const float pi=3.14159;

const char caracter= ‘m’;

const char fecha[]=”25 de diciembre”;

3.13. Caso Especial Constantes Simbólicas

Las constantes simbólicas, se declaran mediante la directiva #define, como se explicó anteriormente. Funcionan de la siguiente manera, cuando C, encuentra el símbolo que representa a la constante, lo sustituye por su respectivo valor.

Ejemplo:

#define N 150

#define PI 3.1416

#define P 50

NOTA: El lector debe comprender algunas diferencias fundamentales entre la declaratoria const y #define; la primera Const, va dentro del programa, es decir, dentro de la función main() o alguna función definida por el usuario, mientras que #define va en el encabezado, después de los #include, por eso estas no llevan al final el punto y coma (;).

3.14. Entrada y Salida Por Consola

Entrada y Salida por consola: se refiere a las operaciones que se producen en el teclado y en la pantalla de la computadora. En C no hay palabras claves para realizar las acciones de Entrada/Salida, estas se hacen mediante el uso de las funciones de la biblioteca estándar (stadio.h).

Para utilizar las funciones de E / S debemos incluir en el programa el archivo de cabecera stdio.h, mediante la declaratoria:

#include <stdio.h>

Las Funciones de E / S más simples son getchar() que lee un carácter del teclado, espera un retorno de carro (¿), es decir un enter y el eco aparece. Es decir la tecla presionada.

*putchar(): Imprime un carácter en la pantalla, en la posición actual del cursor.

Algunas variaciones:

*getche(): Aparece el Eco

*getch(): No aparece el eco

estas instrucciones se encuentran en la biblioteca conio.h

Veamos un ejemplo:

Programa que espera que se presiona una tecla, la muestra en pantalla, y además muestra el carácter siguiente:

Ejemplo 3.1:

```
#include <stdio.h>

#include <conio.h>

main()

{

  char car;
```

```
    car=getchar();
  putchar(car+1);
  getch();
  return 0;
}
```

Ejemplo 3.2:

```
#include <stdio.h>
#include <conio.h>
main()
{
    char x; /*Declaramos x como caracter*/
    printf("Para Finalizar Presione cualquier Tecla:");
    x= getchar();/*Captura y muestra el caracter presionado*/
    getch();/*Espera a que se presione cualquier otra tecla para finalizar*/
    return 0;
}
```

3.15. Entrada / Salida de Cadenas

Una Cadena, es una frase, compuesta por varias palabras. En C, podemos hacer uso de las cadenas, mediante, la sentencia:

*gets(): Lee una cadena de carácter introducido por el teclado. Se puede introducir caracteres hasta que se de un retorno de carro, (enter); el cual no es parte de la cadena; en su lugar se coloca un terminador nulo \0.

*puts(): Imprime en pantalla, el argumento guardado en la variable que se manda a impresión.

Ejemplo:

Diseñe un programa en C, que lea su nombre; lo salude y mande a impresión su nombre, usando gets e y puts

```
#include <stdio.h>

#include <conio.h>

main()

  {

     char nombre[40];

     puts("digite su nombre:");

     gets(nombre);

     puts("BIENVENIDO:");

     puts(nombre);

     getch();

     return 0;

  }
```

3.16. Entrada / Salida Por Consola con Formato

Las funciones gets, puts, getch, etc; son utilizadas, en una forma un poco rudimentaria, sin embargo; C posee otra serie de funciones, que son más completas, las cuales nos permiten leer e imprimir (en pantalla), datos con un formato determinado, el cual ha sido definido por el programador.

Salida Hacia Pantalla [printf()]

Se utiliza para imprimir en pantalla cadenas de texto solas, o mandar a pantalla el valor de alguna variable, o constante, o una combinación de las anteriores. Su formato es el siguiente:

Printf(“cadena de control”, nombre_de_variables);

Cadena de control: contiene códigos de formato que se asocian con los tipos de datos contenidos en las variables.

Código	Formato
%d	Un entero
%i	Un entero
%c	Una caracter
%s	Una cadena
%f	Un real
%ld	Entero largo
%u	Decimal sin signo
%lf	Doble posición
%h	Entero corto
%o	Octal
%x	Hexadecimal
%e	Notación Científica
%p	Puntero
%%	Imprime Porcentaje

Ejemplo:

Int suma=10;

Printf(“La suma es %d”, suma);

Explicación:

Declaramos primero la variable como entero, con un valor de 10, luego la función printf, el mensaje va entre comillas dobles, luego en el lugar que queremos que aparezca el valor, colocamos el formato de la variable, cerramos comillas, luego una coma y el nombre de la variable. Es importante recalcar, que en la posición que coloquemos el formato es donde aparecerá el valor de la variable en este caso, 10.

Ejemplo:

Char nombre[7]=”Manuel”;

printf(“%s es en creador de este manual”, nombre);

NOTA: el número de argumentos que tendrá la función printf() es indefinido, por lo que se puede transmitir cuantos datos sean necesarios.

Ejemplo:

Int x=12, y=15;

char z=’D’;

float v=10.2563;

printf(“Estos son números %d %d %f; y esta es una letra %c”, x,y,v,z);

También podemos hacer algunos arreglos, al formato de salida, por ejemplo, si deseamos imprimir un número real justificado a la izquierda podemos colocar:

printf(“%-f”, z);

para justificar colocarle signo: %+f

%20f >> Longitud numérica del campo

%.2f >>Imprime el valor con sólo dos decimales

3.17. Secuencias de Escapes

Indica que debe ejecutar algo extraordinario.

Carácter de Escape	Explicación
\n	Simula un Enter. Se utiliza para dejar una línea de por medio
\t	Tabulador horizontal. Mueve el cursor al próximo tabulador
\v	Tabulador vertical.
\a	Hace sonar la alarma del sistema
\\	Imprime un carácter de diagonal invertida
\?	Imprime el carácter del signo de interrogación
\”	Imprime una doble comilla

Ejemplos:

1) printf("Manuel \n Antonio \n Ortez\n\n);

2) int x=15;
   printf("El Valor de la variable es %d\n\n", x);

3) float x=8.5689, pi=3.1416;
   printf("El valor de x es %.2f\t\n",x);
   printf("\t Y el valor de pi es %.2f\n\n", pi);

3.18. Entrada Desde Teclado

Se realiza mediante la función scanf(), su formato es:

scanf("Cadena de control", Dirección y nombre de la variable**);**

Ejemplo:

Diseñe un programa que guarde y muestre la nota del examen final de 3 alumnos

```
#include <stdio.h>

#include <conio.h>

main()

  {

     float n1, n2, n3;

     char nom1[10], nom2[10], nom3[10];

     printf("Introduzca el Nombre del Primer alumno:\n");

     scanf("%s", nom1);

     printf("Introduzca la nota de este alumno:\n");

     scanf("%f", &n1);

     printf("Digite el nombre del segundo alumno:\n");

     scanf("%s", nom2);

     printf("Su nota es:\n");

     scanf("%f", &n2);

     printf("Finalmente el ultimo alumno es:\n");
```

```
    scanf("%s", nom3);

    printf("Y su nota es:\n");

    scanf("%f", &n3);

    getch();

    return 0;

}
```

Explicación:

Primero, iniciamos con las directivas del preprocesador:

#include <stdio.h>

#include< conio.h>

Con la cual le indicamos al compilador, que de su librería añada a nuestro programa las funciones estándar de entrada y salida; así como las entradas y salidas por consola (stadio.h y conio.h, respectivamente).

Luego declaramos la variables, que contendrán las notas como reales (o de punto flotante:

float n1, n2, n3;

Ya que, las notas pueden ser decimales, por ejemplo 9.6, 8.5; etc.

Luego declaramos las variables, que contendrán las notas, caba aclarar que al momento de las declaraciones las podemos hacer en el orden que deseemos, pueden ser primeros los tipo char y luego los float, o viceversa, pero teniendo el cuidado que las variables que contendrán las nombres lleven la longitud máxima entre corchetes, para nuestro caso, 10. ([10]).

Posteriormente, mostramos en pantalla, un mensaje con el cual le indicamos al usuario que introduzca los datos respectivos:

printf("Introduzca el Nombre del Primer alumno:\n");

A continuación, va la función scanf, primero y entre comillas el tipo de dato que va a leer:

scanf("%s", nom1);

como puede notarse, va a leer la cadena de texto que contendrá la variable nom1. cabe aclarar, que cuando se van a leer cadenas de texto, no es necesario colocar la dirección (&), lo cual no sucede con los otros tipos de datos:

```
scanf("%f",& n1);
```

Después de haber leído los datos, espera a que se presiones cualquier tecla para finalizar la ejecución del programa.

3.19. OPERACIONES BASICAS

Operadores

Un operador, es un símbolo que indica al compilador que se lleve a cabo ciertas manipulaciones matemáticas o lógicas.

Operadores Aritméticos

Operador	Propósito
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de la división entera

Todos estos operadores se pueden aplicar a constantes, variables y expresiones. El resultado es el que se obtiene de aplicar la operación correspondiente entre los dos operandos. (Tomado de "Aprenda Lenguaje ANSII C, como si estuviera en primero". Pag. 25).

Los operandos sobre los que actúan los operadores aritméticos deben ser valores **Numéricos,** es decir datos enteros, punto flotante o de carácter (Int, float y char, respectivamente).

Una aclaración especial, merece el operador "%", que indica el resto de la división entera. Veámoslo con un ejemplo:

Si dividimos 30/3, su cociente es 10, y su residuo es 0. Si dividimos 25/3, su cociente es 8, y tiene un residuo de 1. Entonces de lo que se encarga, este operador, es de devolvernos el valor del residuo de una división. Cabe aclarar que los datos deben de ser tipo entero, y su sintaxis es la siguiente:25%3

NOTA: Este Operador, NO puede aplicarse a los datos de tipo float.

Una **Expresión,** Es un conjunto de variable, constantes y otras expresiones más sencillas, relacionadas por algún tipo de operador. De las cuales hablaremos con más detalle, posteriormente.

OPERADORES RELACIONALES O DE COMPARACIÓN:

Operador	Significado
<	Menor que
<=	Menor o igual que

>	Mayor que
>=	Mayor o igual que
==	Igual que (Para las comparaciones)
!=	No igual a

3.20. EXPRESIONES

(Tomado de "Aprenda ANSII C como si estuviera en Primero", Universidad de Navarra. 1998).

Ya han aparecido algunos ejemplos del lenguaje C en las secciones precedentes. Una Expresión es una combinación de variables y/o constantes, y operadores. La expresión es equivalente al resultado que proporciona al aplicar sus operadores a sus operandos. Por ejemplo 1 + 5 es una expresión formada por dos operandos (1 y 5)y el operador (el +); esta expresión es equivalente al valor 6, por lo cual quiere decir que allí donde esta expresión aparece en el programa, en el momento de la ejecución es evaluada y sustituida por su resultado. Una expresión puede estar formada por otras expresiones más sencillas, y puede contener paréntesis de varios niveles agrupando distintos términos. En C, existen diferentes tipos de expresiones. El cual depende del tipo de operadores que se estén utilizando. Por ejemplo: Expresiones lógicas, aritméticas, etc

Se debe hacer hincapié en que, si existen algunas expresiones encerradas entre paréntesis, estas se evalúan primero. Ejemplo:

9*(8+5)

primero sumamos 8+5, cuyo resultado es 13, y este lo multiplicamos por nueve, con lo que la expresión anterior, da cómo resultado: 117.

Si existen expresiones en paréntesis anidadas, es decir, que uno se encuentra dentro de otros paréntesis, se evalúan los más internos. Ejemplo:

2*((20/(12-2))+5)

se evalúa la operación 12-2, que da como resultado 10, luego se divide 20, entre el resultado anterior, es decir 10. el resultado es 2, y a este número se le suma 5, obteniendo 7. ahora se multiplica por dos, para determinar así que la expresión anterior es igual a 14.

3.21. Estructuras Secuenciales

Se les denomina así, por qué; son estructuras en un programa, que después de ejecutar una instrucción o sentencia, continúan con la otra, hasta llegar al final del programa. Los

ejemplos que hemos visto anteriormente, son ejemplos de estructuras secuenciales. Veamos otros ejemplos:

Ejemplo 3.3

Diseñe un programa que calcula el cuadrado y el cubo de tres números introducidos por el usuario.

```
#include <stdio.h>

#include <conio.h>

main()

{

int x, x1, x2, y, y1, y2, z, z1, z2;

clrscr();

printf("\tPROGRAMA QUE CALCULA EL CUADRADO Y EL CUBO DE 3 NUMEROS\n\n");

printf("Introduzaca el primer n£mero:\n");

scanf("%d", &x);

printf("Ahora ingrese el siguiente n£mero:\n");

scanf("%d", &y);

printf("Y el tercer n£mero es:\n");

scanf("%d", &z);

x1=x*x;

x2=x*x*x;

y1=y*y;

y2=y*y*y;

z1=z*z;

z2=z*z*z;
```

```
printf("**********************************\n");

printf("**Numero****Cuadrado*****Cubo****\n");

printf("**%d **** %d ***** %d ****\n", x, x1, x2);

printf("**%d **** %d ***** %d ****\n", y, y1, y2);

printf("**%d **** %d ***** %d ****\n", z, z1, z2);

printf("**********************************\n");

getch();

return 0;

}
```

Ejemplo:

Una empresa necesita conocer el sueldo neto a pagar a un empleado. Teniendo como entrada el salario produzca una salida de sueldo neto. Los descuentos a aplicar son: ISSS 5%, AFP 7% y Renta 10%, estos descuentos son sobre el salario, y es sueldo neto es la diferencia entre el salario y el total de las retenciones:

```
#include <stdio.h>

#include <conio.h>

main()

{

float sueldo, afp, isss, renta, sn;

char nombre[50];

clrscr();

printf("Introduzca el Nombre del empleado:\n");

scanf("%s", nombre);

printf("Su sueldo es:\n");

scanf("%f", &sueldo);
```

```
afp=sueldo*0.07;

isss=sueldo*0.05;

renta=sueldo*0.10;

sn=sueldo-(afp+isss+renta);

printf("El empleado %s\n", nombre);

printf("Posee un sueldo neto de %.2f\n", sn);

getch();

return 0;

}
```

Ejemplo:

Diseñe un programa que calcule el promedio y la suma de tres números ingresados por el usuario:

```
#include <stdio.h>

#include <conio.h>

main()

{

float x, y, z, sum, prom;

clrscr();

printf("El Primer n£mero es:\n");

scanf("%f", &x);

printf("Ahora el segundo n£mero:\n");

scanf("%f", &y);

printf("El Ultimo numero es:\n");

scanf("%f", &z);
```

```
sum=x+y+z;

prom=sum/3;

printf("*****************************************\n");

printf("**La suma es %.2f y el promedio es %.2f*\n", sum, prom);

printf("*****************************************\n");

getch();

return 0;
```

3.22. Estructuras Selectivas

Los pequeños programas que hemos diseñada hasta el momento, han sido del tipo secuencial, es decir, una sentencia se ejecuta después de otra, hasta el final del programa.

Pero en la vida diaria muchas veces debemos elegir entre un camino y otro para llegar a nuestro destino. Lo mismo pasa en programación, al realizar alguna actividad, nuestro programa debe ser capaz de elegir uno u otro camino, a seguir dependiendo del valor de alguna condición evaluada.

Para ello C, dispone de tres tipos de 3 tipos de estructuras selectivas, la cuales son:

- Estructura Selectiva Simple
- Estructura Selectiva Doble
- Estructura Selectiva Múltiple

3.23. ESTRUCTURA SELECTIVA SIMPLE

Funciona de la siguiente manera: se evalúa una condición, de ser cierta efectúa una acción, de lo contrario, continúa con la ejecución normal del programa.

Su sintaxis es la siguiente:

If(condición) Acción;

O también:

If(Condición)

Acción;

Donde:

Condición: Es una expresión lógica que es evaluada por el compilador

Acción: es la Acción o Acciones que realizará el programa de resultar cierta la condición

NOTA: En C, no existe la sentencia "End If", como en otros lenguajes de programación para indicar que ha terminado el bloque de selección, sino que este se especifica con el punto y coma al final. Además que, después de la condición NO se escribe un punto y coma. Si son varias acciones, estas deben ir dentro de llaves {}, para indicarle al compilador que son un solo bloque de acciones que deben ejecutarse.

Ejemplo:

En una tienda se venden artículos de primera necesidad, a los cuales se les aplica un descuento del 20%, de la compra total, si esta es igual o mayor a $50. Diseñe un programa en C, que a partir del importe total de la compra muestre lo que debe pagar el cliente.

```
#include <stdio.h>

#include <conio.h>

main()

  {
     float compra;

     clrscr();

     printf("Introduzca el valor de la compra:\n");

     scanf("%f", &compra);

     if(compra>=50)

           compra=compra*0.8;

      printf("El Importe de la compra es %.2f\n\n", compra);

     getch();
```

```
    return 0;

}
```

3.24. ESTRUCTURA SELECTIVA DOBLE

Esta estructura, se caracteriza por el hecho que ofrece dos caminos a seguir, dependiendo si al evaluar la condición resulta cierta o falsa. Su sintaxis es la siguiente:

```
if(Condición)

    Acción 1;

else

    Acción 2;
```

Funciona, de la siguiente manera si condición, al evaluarla resulta cierta, realiza la acción 1. de lo contrario, es decir; si al evaluar la condición resulta falsa, realiza la acción 2.

Se debe tener en cuenta la condición puede ser compuesta, es decir haciendo uso de los operadores && y || (Y lógico y No lógico), además que cuando tenemos más de una sentencia por ejecutar ya sea del lado del cierto o del falso, estas van dentro de llaves.

Ejemplo:

Se desea saber si un número es par o impar. Diseñe un programa en el cual el usuario, ingrese el número y el programa muestre con un mensaje, si éste es par o no.

```
#include <stdio.h>

#include <conio.h>

main()

  {
int num;
    printf("Ingrese el n£mero:\n");

scanf("%d", &num);

    if(num%2==0)

printf("ES PAR\n\n");
```

```
    else
        printf("ES IMPAR\n\n");
    getch();
    return 0;
}
```

3.25. Entornos de Desarrollo

3.25.1. DEV C++:

- **Dev-C++** es un entorno de desarrollo integrado (IDE) para programar en lenguaje C/C++. Usa MinGW, que es una versión de GCC (GNU Compiler Collection) como su compilador. Dev-C++ puede además ser usado en combinación con Cygwin y cualquier compilador basado en GCC.
- El Entorno está desarrollado en el lenguaje Delphi de Borland. Tiene una página de paquetes opcionales para instalar, con diferentes bibliotecas de código abierto.

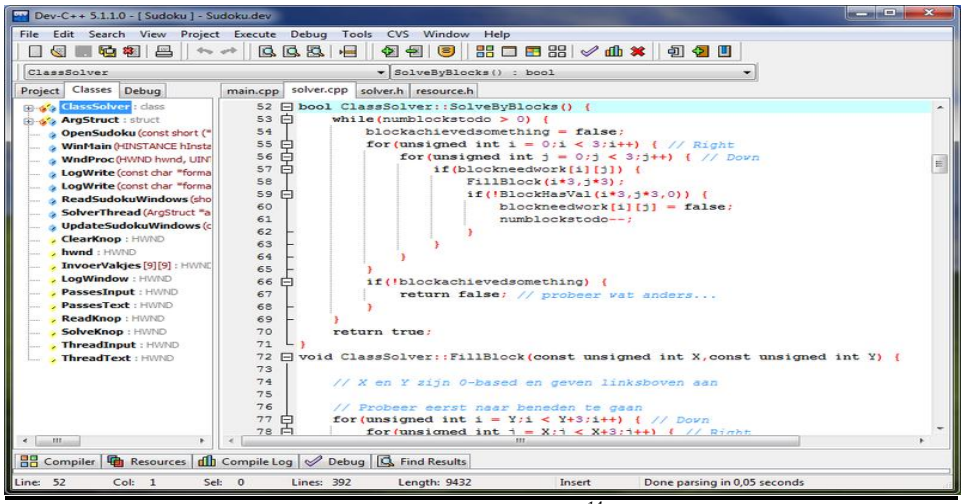


Figura 10[14]

Algunas de las características de Dev-C++ son:

- Soporta compiladores basados en GCC, por ejemplo Mingw.
- Tiene integrado un depurador basado en GDB (Gnu DeBugger).
- Mantiene una lista con las clases utilizadas durante la edición de un programa.
- Mantiene una lista de las funciones definidas en la implementación del programa.

[14] https://upload.wikimedia.org/wikipedia/commons/thumb/b/b9/Devcpp5110.png/800px-Devcpp5110.png

- Tiene un manejador de proyectos.
- Soporta la actualización del software y bibliotecas a través de Internet.

3.26. EJEMPLOS

Ejemplo:

Diseñe un programa, que dada la nota de alumno, imprima en la pantalla un comentario sobre esa nota. El criterio para los comentarios es el siguiente:

Si nota es mayor o igual a 9 “Excelente”

Si nota es mayor o igual a 8 “Muy Bueno”

Si nota es mayor o igual a 7 “Bueno”

Si nota es mayor o igual a 6 “Regular”

Si nota es menor que 6 “Necesita Mejorar”

```
#include <stdio.h>

#include <conio.h>

main()

  {
    float nota;
    printf("Digite la nota, porfavor:\n");
    scanf("%f", &nota);
    if(nota >= 9.0)
printf("EXCELENTE\n\n");
    else
         if(nota >= 8.0)
printf("MUY BUENO\n\n");
else
           if(nota >= 7.0)
             printf("BUENO\n\n");
           else
if(nota >=6.0)
                printf("REGULAR\n\n");
             else
                printf("NECESITA MEJORAR\n\n");
```

```
    getch();

    return 0;

  }
```

Este ejemplo, muestra que C, permite hacer anidamientos, es decir, una selección dentro de otra, ya sea del lado del cierto, o del falso o de ambos.

Ejemplo:

Diseñe un programa en C, que dado un número del 1 al 3, muestre en pantalla y en letras, el mismo número:

```
#include <stdio.h>
#include <conio.h>
main()
  {
    int n;
    clrscr();
    printf("El N£mero es:\n");
    scanf("%d",& n);
    switch(n)
        {
          case 0: puts("Cero");
                  break;
          case 1: puts("Uno");
                  break;
          case 2: puts("Dos");
                  break;
          case 3: puts("Tres");
                  break;
          default: puts("Dato No valido");
                  break;
        }
    getch();
    return 0;
  }
```

3.27. EJERCICIOS PROPUESTOS

1. ¿Cuál es la salida del siguiente programa?

```
void main ()
```

```
{
float x, y;

printf ("Introduzca 2 números:\n");
scanf ("%f%f", &x, &y);
printf ("La suma de %f y %f vale %f\n",  x,  y,  x+y);
printf ("La suma de %4f y %4.2f vale %10.3f\n",  x,  y,  x+y);
printf ("La suma de %e y %e vale %e\n",  x,  y,  x+y);
}
```

2. Escribir un programa que lea un número entero, lo multiplique por dos y a continuación lo escriba por pantalla.
3. Escribir un programa que solicite la cantidad de artículos disponibles en un comercio y su precio. El programa debe mostrar el valor de las existencias del comercio.
4. Escribir un programa que calcule la mitad, el cuadrado y el cubo de un número real solicitado al usuario. Se puede utilizar la función pow(a,b) que calcula a b
. Esta función está incluida en la librería math.h.
5. Escribir un programa que solicite dos valores enteros al usuario e intercambie el contenido de las dos variables leídas.
6. Escribir un programa que lea dos números enteros A y B, y obtenga los valores A dividido por B y su resto.
7. Escribir un programa que solicite la longitud y anchura de una habitación y muestre su superficie.
8. Calculo del área de un círculo ($A=\pi r_2$). Pedir el radio del círculo al usuario.
9. Escribir un programa que convierta un número de segundos en su equivalente en días, horas, minutos ysegundos. Debe visualizarse dd - hh:mm:ss.
10. ¿Cuál es la salida por pantalla de cada uno de éstos programa?

```
void main ()
{
int a, b, c, d=6, e;

a = b = 3;
c = a*b+d;
e = (c + 5) / 4-3;
e+=5;
printf("Los resultados son %d y %d ", c, e);
}
void main ()
{
int a, b, c, d=6, e;

a = 3;
b = a - d / 3;
a *=b;
c = a + d / a - 3 / a * b;
e = c + 8 / 4-b;
e+=5;
printf("Los resultados son %d y %d ", c, e);
}
```

11.Haciendo uso de las funciones gets y puts, diseñe un programa en C, que se lea el nombre del usuario y lo muestre en pantalla junto con un saludo.

12. Diseñe un programa en C, que lea y muestre en pantalla el valor de tres variables de tipo Entero.

3.28. EJERCICIOS RESUELTOS

Ejercicio:

Se desea saber si un número es par o impar. Diseñe un programa en el cual el usuario, ingrese el número y el programa muestre con un mensaje, si éste es par o no.

```
#include <stdio.h>

#include <conio.h>

main()

  {
        int num;

     printf("Ingrese el n£mero:\n");

     scanf("%d", &num);

     if(num%2==0)

printf("ES PAR\n\n");

     else

           printf("ES IMPAR\n\n");

     getch();

     return 0;

  }
```

Ejercicio:

Diseñe un programa, que dada la nota de alumno, imprima en la pantalla un comentario sobre esa nota. El criterio para los comentarios es el siguiente:

Si nota es mayor o igual a 9 “Excelente”

Si nota es mayor o igual a 8 “Muy Bueno”

Si nota es mayor o igual a 7 “Bueno”

Si nota es mayor o igual a 6 “Regular”

Si nota es menor que 6 “Necesita Mejorar”

```
#include <stdio.h>
#include <conio.h>
main()
  {
    float nota;
    printf("Digite la nota, porfavor:\n");
    scanf("%f", &nota);
    if(nota >= 9.0)
printf("EXCELENTE\n\n");
    else
          if(nota >= 8.0)
printf("MUY BUENO\n\n");
else
             if(nota >= 7.0)
               printf("BUENO\n\n");
             else
if(nota >=6.0)
                    printf("REGULAR\n\n");
               else
```

```
                printf("NECESITA MEJORAR\n\n");

    getch();

    return 0;

  }
```

Ejercicio:

Dada el peso, la altura y el sexo, de unos estudiantes. Determinar la cantidad de vitaminas que deben consumir estos estudiantes, en base al siguiente criterio:

- Si son varones, y su estatura es mayor a 1.60, y su peso es mayor o igual a 150 lb, su dosis, serán: 20% de la estatura y 80% de su peso. De lo contrario, la dosis será la siguiente: 30% de la estatura y 70% de su peso.

- Si son mujeres, y su estatura es mayor de a 1.50 m y su peso es mayor o igual a 130 lb, su dosis será: 25% de la estatura y 75% de su peso. De lo contrario, la dosis será: 35% de la estatura y 65% de su peso. La dosis debe ser expresada en gramos.

```
#include <stdio.h>
#include <conio.h>
main()
  {
    float peso, estatura, dosis;
    char sexo;
    printf("Introduzca el sexo del alumno(a)<H/M>:\n");
    scanf("%c", &sexo);
    printf("Peso:\n");
    scanf("%f", &peso);
    printf("La estatura es de:\n");
scanf("%f", &estatura);
    if(sexo=='H' || sexo=='h')
{
            if(estatura>1.60 && peso >=150)
                {
                 dosis=(0.20*estatura)+(0.8*peso);
                 printf("La dosis de este alumno ser : %.2f gramos\n\n", dosis);
                }
            else
                {
                 dosis=(0.3*estatura)+(0.7*peso);
                 printf("La dosis de este alumno sera %.2f gramos\n\n", dosis);

                }
```

```
        }
    else
        {
          if(estatura>1.50 && peso >=130)
             {
               dosis=(0.25*estatura)+(0.75*peso);
               printf("La dosis de esta alumna debe ser de %.2f gramos\n\n", dosis);
             }
          else
             {
               dosis=(0.35*estatura)+(0.65*peso);
               printf("La dosis de esta alumna debe ser de %.2f gramos\n\n", dosis);
             }
        }

    getch();

    return 0;

}
```

Capítulo No 4:

4.1. CICLOS

Para repetir varias veces un proceso determinado haremos uso de los ciclos repetitivos, a los cuales se les conoce con el nombre de estructura repetitiva, estructura iterativa, lazo o bucle.

(Tomado de Los guiones de clase de Introducción a la Informática. Universidad de El Salvador. Año 2005)

En C, podemos encontrar tres tipos de ciclos:

· Entrada Asegurada (while)

· Ciclo Controlado Por Contador (for)

· Hacer Mientras (do.. while)

4.2. Funcionamiento de Un Ciclo

Un ciclo, funciona de la siguiente manera: Evalúa una condición de resultar cierta, realiza una acción o bloque de acciones, luego vuelve a evaluar la condición y si nuevamente resulta cierta, realiza la (s) acción (es). Cuando la condición de cómo resultado falso, se sale del ciclo y continúa con la ejecución normal del programa.

Acumulador:

Es una variable, que , como su nombre lo indica se encarga de acumular valores. Esto se vuelve muy útil, por ejemplo, cuando queremos encontrar la suma de los números del 0 al 9, en el acumulador, vamos guardando los valores de dichas cifras. Puede ser tanto real como entera. Su valor inicial, en la mayoría de los casos es cero.

Contador:

Es una variable de tipo entero, que nos ayuda, en el programa a contabilizar el número de ejecuciones de una misma acción, de un grupo de alumnos etc. Un acumulador tiene tres valores distintos:

· Valor Inicial: es el valor con el cual iniciamos nuestro contador. Generalmente es cero. Esta asignación puede hacerse cuando se declara la variable.

· Valor Final: después de la ejecución del ciclo, el valor del contador, será distinto a su valor inicial, este puede ser mayo o menor que el mismo, todo depende si fue una cuenta creciente o decreciente.

· Valor de Cambio: Es el valor Constante, en el cual se irá incrementando nuestro contador, este puede ser positivo o negativo; es decir, si la cuanta se realiza de manera ascendente o descendente.

NOTA: el lector no debe confundirse entre las variables tipo acumulador y tipo contador, estas se diferencian unas de otras en que: los contadores, su valor de cambio es una constante, ya que aumenta y disminuyen en el mismo valor, mientras que los acumuladores su valor de cambio no es constante. Un acumulador necesariamente lo inicializamos con cero (o al menos en la mayoría de los casos). Un contador puede iniciar con cualquier valor.

Bandera:

Las variables tipo bandera son aquellas que sólo admiten dos valores: cierto o falso, true o false, hombre o mujer... etc

4.3. Ciclo de Entrada Asegurada

La sintaxis es la siguiente:

while(condición)

Acción;

Funciona de la siguiente manera: primero evalúa la condición, si da como resultado cierta realiza la acción, luego vuelve a evaluar la condición, si su resultado es falso, se sale del ciclo y continúa con la ejecución del programa.

Hay que tener mucho cuidado, cuando trabajamos con ciclos, ya que podemos caer en un ciclo infinito, es decir que nunca se sale de él. Lo cual no es un error de sintaxis sino de lógica. Por lo cual en las acciones debemos siempre colocar algo que haga que se modifique el resultado de la condición, lo cual puede ser una bandera, un contador o un acumulador.

Ejemplo :

Diseñe un Programa que imprima los primeros 10 números.

```
#include <stdio.h>

#include <conio.h>

main()

  {

     int i=1; /*Declaramos nuestro contador con su Valor Inicial*/

     while(i<=10) /*Mientras i sea menor o igual a 10:*/

          {

            printf("%d\t", i);/*Imprimir el valor de i*/

            i+=1;/*Aumentar el contador en 1*/

          }

     getch();

     return 0;

  }
```

4.4. Ciclo Controlado por contador.

En algunas ocasiones, sabemos a ciencia cierta el número de veces que se tiene que repetir una misma acción o bloque de acciones. Y para ello es que nos sirve, esta estructura. Su sintaxis es la siguiente:

```
for( valor inicial; condición; incremento)

    accion;
```

Donde:

Valor inicial: es el valor con el cual inicializamos nuestra variable de control.

Condición: si la cumple, ejecuta la acción o acciones e incrementa o decrementa la variable de control, sino la cumple la condición, se sale del ciclo.

Incremento; que puede ser positivo o negativo (decremento).

Veamos un ejemplo sencillo:

Ejemplo 4.4:

Diseñe un programa que imprima los primeros 10 números:

```
#include <stdio.h>

#include <conio.h>

main()

  {

    int i;

    for(i=1; i<=10; i++)

          printf("%d\t", i);

    getch();

return 0;

  }
```

4.5. CONVERSION DE TIPOS

En C, existe, además, de la conversión automática de tipos de datos, laposibilidad de forzar la conversión de un tipo de datos en otro tipo de datos. Estaconversión de un tipo de datos en otro se llama “casts”, y su sintaxis es:

(tipo)expresión
Su utilidad queda claramente expresada en el ejemplo siguiente:

```
int a=3,b=2;
float c;
c=a/b;
```

La operación asigna a c el valor 1.0 en vez de el valor 1.5, ello se debe a que alser a y b variables de tipo entero, se realiza una división entre enteros, y el resultado de3/2 es 1. A continuación ese valor 1 se convierte a un valor en coma flotante pararealizar la asignación (valor 1.0), y se asigna a c. Si lo que se desea es que la divisiónse realice en punto flotante, debe escribirse la operación de la siguiente forma:

```
c=(float)a/b;
```

Esta conversión forzada obliga a convertir la variable a en float, y entonces,aplicando las reglas de conversión automática de tipos, se realiza la división en comaflotante. Este proceso da lugar a que el resultado de la operación sea 1.5, y dicho valorsea el asignado a la variable c.

4.6. Ejercicios Propuestos:

Ejercicios
1.Se desea conocer la suma de los números enteros, positivos menores que n, el cual es un dato dado por el usuario.
2.Muestre un programa en c, que imprima en pantalla los números desde un valor inicial, hasta un valor final, ingresados por el usuario, tanto en forma descendente como ascendente.
3.Diseñe un programa que imprima la serie de Fibonacci, así: 0 1 1 2 3 5 8 13.... hasta un número n dado por el usuario.
4.Calcule el promedio de edades de un grupo de estudiantes, de los cuales no se conoce la cantidad.
5.Diseñe un programa que obtenga, la calificación mayor y la calificación menor, de un grupo de 40 estudiantes, además de los nombres de dichos alumnos.
6.En un país hubieron elecciones para elegir al presidente. El país consta de 7 provincias o regiones, de las cuales se han levantado actas que contiene el total de votos obtenidos por los 4 partidos políticos en dicha región. Diseñe un programa en c, que lea las actas de las 7 provincias, muestre que partido ganó las elecciones y en caso de empate, lo especifique con un mensaje.
7.en un supermercado, hay 3 departamentos (de ropa, comestibles y perfumería), en lo cuales se realizan un descuento de 5%, 3.5% y 8% respectivamente, por las compras totales mayores de $100.00. diseñe un programa que dado el monto de la compra, realice los descuentos pertinentes por departamento, le indique al usuario a cuanto asciende su nuevo monto e indique, cuanto fue lo recaudado al final del día.
8.La Empresa, el porvenir s.a de c.v desea conocer lo que debe pagar en concepto de horas extras aun grupo de n empleados. Se sabe que una hora extra diurna, se paga el doble que una hora normal. Y una hora extra nocturna se paga el doble de una hora normal más el 25%. Además que todos los empleados tiene sueldos diferentes, muestre el nuevo sueldo de cada uno de ellos y lo que tendrá que pagar la empresa en concepto de horas extra.

9.Una compañía de teléfonos, cobra $0.03 por minuto la llamada nacional local, $0.06 por la llamada de larga distancia nacional y $0.10 la llamada de larga distancia internacional. Diseñe un programa que calcule las facturas mensuales de los clientes, sabiendo que, si las llamadas fueron realizadas por la mañana tienen un doble valor, y si los 10 primeros minutos de llamadas locales son gratis, en cualquier horario.

4.7. Ejercicios Resueltos

Ejercicio:

Se desea conocer el promedio de los números mayores que cero, en una serie de números ingresados por el usuario. De los cuales no se sabe la cantidad, haciendo uso de una bandera, diseñe un programa en el cual el usuario ingrese los números que desee.

```
#include <stdio.h>

#include <conio.h>

main()

  {

     int i=0, sum=0, ban=1, n;

     float prom;

     while(ban==1)

          {

             printf("Ingrese un n£mero por Favor:\n");

             scanf("%d", &n);

             if(n>0)

                  {
                     i=i+1;
sum+=n;
                  }

             printf("Desea Ingresar Otro N£mero? (Si=1 y No=0)\n");

             scanf("%d", &ban);
}

     prom=sum/i;
```

```
    printf("*************************************************************\n");

printf("*** El Promedio de los numeros mayores que cero es: %.2f ***\n", prom);

printf("*************************************************************\n");

    getch();

    return 0;

  }
```

Ejercicio:

En un salón se tienen las notas de 14, alumnos; de los cuales se desea saber cual fue el promedio de todas las notas, cual fue la nota mayor y la nota menor. Así como la cantidad de aprobados en el curso (Para Aprobar la asignatura se requiere de una nota mayor o igual a 6.0)

```
#include <stdio.h>

#include <conio.h>

main()

  {

     float suma=0, prom, menor=11, mayor=-1, nota;
    int i=1,j=0;
    while(i<=14)

{
          printf("Ingrese la Nota del alumno %d:\n", i);

          scanf("%f", &nota);

          while(nota<0.00 || nota >10.00)

            {
                printf("ERROR, la nota debe estar entre 0 y 10\n");
                scanf("%f", &nota);
```

```
            }

          if(nota>=6.00)

            j=j+1;

          if(nota>mayor)
            mayor=nota;
          if(nota<menor)
            menor=nota;
          i=i+1;
          suma=suma+nota;
        }

    prom=suma/14;

    printf("El Promedio es %.2f\n\n", prom);

    printf("El total de Aprobados es %d\n", j);

    printf("La Mayor nota fue %.2f\n", mayor);

    printf("%.2f corresponde a la nota menor\n", menor);

    getch();

    return 0;

  }
```

Ejercicio:

Diseñe un programa en C, que calcule las compras totales, realizadas por un grupo de 20 amas de casa. Luego con esa información obtenga la media.

```
#include <stdio.h>
#include <conio.h>
main()
  {
        int i;
        float compra, desvia, prom, varinza, sum=0;
        for(i=1; i<=10; i++)
          {
              printf("Ingrese la cantidad que gast¢ la ama de casa %d:\n", i);
              scanf("%f",& compra);
              while(compra<0)
```

```
                    {
                      printf("ERROR, la compra debe ser mayor que cero, vuelva a
intentarlo:\n");
                      scanf("%f", &compra);
                    }
                  sum=sum+compra;
              }
          prom=sum/12;
          printf("El promedio de las compras es %.2f\n\n\a", prom);
          getch();
          return 0;
    }
```

4.8. Ciclo Do... while

Es te ciclo funciona de la siguiente manera, realiza la acción o conjunto de acciones, luego evalúa una condición de resultar cierta vuelve a realizar la/s accion/es. Cuando sea falsa, se sale del ciclo. Esta estructura, no está lógicamente, estructurada, por ello, no hablaremos mucho, sin embargo realizaremos un par de ejemplos, de este ciclo.

Formato :

```
        do {

            sentencia;

            .

            .

             } while(<expL>);
```

La diferencia fundamental, entre el ciclo while y do...while, es que en este ultimo, las sentencias se realizarán por lo menos una vez, en cambio, con while, solo se cumplirán mientras se cumpla la condición, lo cual puede ser nunca.

Ejemplo :

Programa que determina si un año es bisiesto o no. Y un año es bisiesto si es múltiplo de cuatro, pero excluyendo aquellos que son múltiplos de 100 pero no de 400

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int anio;
  char respuesta;
```

```
  printf("\n\n\nINICIO DEL PROGRAMA\n\n\n");
  printf("\n\nEl programa te pide un anio y te dice exactamente si es bisiesto o no");
  do
  {
  /*ENTRADA DE DATOS*/
    printf("\n\nIntroduzca un anio determinado ");
    scanf("%d",&anio);
  /*PROCESO Y SALIDA DE DATOS*/
    if ((anio%4==0 && anio%100!=0)||(anio%400==0)) printf("\n\nEl anio es bisiesto");
    else printf("\n\nEl anio no es bisiesto");
    printf("\n\nDesea introducir mas datos\n\n");
    respuesta=getch();
  } while(respuesta=='S' || respuesta=='s');
  printf("\n\n\nFIN DEL PROGRAMA\n\n\n");
}
```

NOTA: este código ha sido tomado de "Practicas de Programación en C", de Fernando Muñoz Ledesma. Practica 3, ejercicio 5.

Capítulo No 5: Funciones

5.1. Funciones en C

La modularización, es una técnica usada por los programadores para hacer sus códigos más cortos, ya que consiste en reducir un gran problema complejo, en pequeños problemitas más sencillos, concentrándose en la solución por separado, de cada uno de ellos.

En C, se conocen como funciones *aquellos trozos de códigos utilizados para dividir un programa con el objetivo que, cada bloque realice una tarea determinada.*

En las funciones juegan un papel muy importe las variables, ya que como se ha dicho estas pueden ser locales o globales.

Variables Globales: Estas se crean durante toda la ejecución del programa, y son globales, ya que pueden ser llamadas, leídas, modificadas, etc; desde cualquier función. Se definen antes del main().

Variables Locales: Estas, pueden ser utilizadas únicamente en la función que hayan sido declaradas.

La sintaxis de una función es la siguiente:

Tipo_de_datos nombre_de_la_funcion(tipo y nombre de argumentos)

{

acciones

}

donde:

- **Tipo_de_datos:** Es el tipo de dato que devolverá esa función, que puede ser real, entera, o tipo void(es decir que no devolverá ningún valor).
- **Nombre_de_la_funcion:** Es el identificador que le damos a nuestra función, la cual debe cumplir las reglas que definimos en un principio para los identificadores.
- **Tipo y nombre de argumentos:** son los parámetros que recibe la función. Los argumentos de una función no son más que variables locales que reciben un valor. Este valor se lo enviamos al hacer la llamada a la función. Pueden existir funciones que no reciban argumentos.
- **Acciones:** Constituye el conjunto de acciones, de sentencias que cumplirá la función, cuando sea ejecutada. Entre ellas están:
    1. Asignaciones
    2. Lecturas
    3. Impresiones
    4. Cálculos, etc

Una función, termina con la llave de cerrar, pero antes de esta llave, debemos colocarle la instrucción **return,** con la cual devolverá un valor específico. Es necesario recalcar que si la función no devuelve ningún valor, es decir, es tipo void, no tiene que ir la sentencia return, ya que de lo contrario, nos dará un error.

5.2. ¿Cómo es que funcionan los Subprogramas?

A menudo, utilizamos el adjetivo de “Subprogramas”, para referirnos a las funciones, así que, el lector debe familiarizarse también con este término.

Los subprogramas se comunican con el programa principal, que es el que contiene a las funciones, mediante parámetros, que estos pueden ser: Parámetros Formales y Parámetros Actuales.

Cuando se da la comunicación los parámetros actuales son utilizados en lugar de los parámetros formales

5.3. Paso de Parámetros

Existen dos formas de pasar parámetros, las cuales son:

A) Paso por Valor

También conocido como parámetros valor. Los valores se proporcionan en el orden de cálculos de entrada.

Los parámetros se tratan como variables locales y los valores iniciales se proporcionan copiando los valores de correspondientes argumentos.

Los parámetros formales-Locales de una función reciben como inicilaes los valores de los parámetros actuales y con ellos se ejecutan las acciones descritas en el subprograma.

Ejemplo:

A=5;

B=7;

C=proc1(A, 18, B*3+4);

Proc1(X, Y, Z)

Explicación:

Donde, se encuentra c, se está llamando la función, denominada proc1, en la cual se están enviando como parámetros el valor de A, que es cinco; el cual es recibido por la variable X, en la definición de la función proc1; en la misma función, Y tendrá el valor de 18; por que ese es el valor del parámetro formal, mientras que Z, tendrá un valor inicial de 25, ya que ese es el resultado del tercer parámetro que resulta ser una expresión aritmética.

5.4. Funciones Definidas Por El Usuario en C

Una función, como ya se ha dicho, es un bloque de código dentro del programa que se encarga de realizar una tarea determinada. Por lo tanto un programa en c debe constar de una o más funciones, y por supuesto no puede faltar la función principal main().

Un viejo adagio dice: Separa y vencerás, lo cual se acopla perfectamente cuando tenemos un programa que es bastante grande; podemos separarlos en pequeños subprogramas (funciones), y concentrarnos en la solución por separados de cada uno de ellos y así resolver un gran problemas, en unos cuantos problemitas más pequeños.

Si un programa, está constituido por más de una función, las llamadas a la misma, pueden realizarse desde cualquier parte del programa, y la definición de ellas debe ser independiente unas de otras.

Por lo tanto sería un grave error el tratar de definir una función dentro de otra.

Una función puede ser llamada desde cualquier parte del programa no sólo una vez, y cuando es llamada, empieza a ejecutar las acciones que están escritas en código.

Para mayor comodidad del lector vamos a ver varios ejemplos, del uso de funciones y a medida que vayamos avanzando se volverán más complejos.

El orden será el siguiente:

1. Funciones que no devuelven ningún valor

2. Funciones que devuelven un valor entero

3. Funciones que devuelven un valor Real

4. Funciones combinadas

5. Funciones en las que usamos Menú.

5.5. Funciones que no devuelven ningún valor.

Cómo se ha dicho las funciones pueden o no devolver algún valor, para mi parecer, este tipo de funciones son las más sencillas, ya que cuando se llama la función, esta realiza lecturas, asignaciones, cálculos o impresiones, finaliza la ejecución de la función y el programa continúa normalmente.

Ejemplo: Diseñe un programa que dados dos números enteros determine la suma y cual de ellos es mayor, usando dos funciones diferentes.

```
#include <stdio.h>

#include <conio.h>

void suma (int a, int b);   /*Declaraci¢n de la funci¢n*/

void mayor (int a, int b);   /*Tipo de dato, nombre de la funci¢n y el tipo y nombre de los argumentos*/

main()

  {

      int a, b;

      printf("Ingrese el valor de a:\n");

      scanf("%d", &a);

      printf("Ingrese el valor de b:\n");

      scanf("%d", &b);

      suma(a,b);  /*Llamado de la funci¢n*/

      mayor(a,b); /*Unicamente el nombre de la funci¢n y de los par metros*/

      getch();

      return 0;
```

```
}

void suma(int a, int b) /*Definici¢n de la funci¢n*/

{                    /*Abrimos llaves al inicio de la definici¢n*/

   int sum;          /*Declaraci¢n de las variables locales*/

   sum=a+b;

   printf("El valor de la suma es %d:\n\n", sum);

}                    /*Fin de la funci¢n suma*/

void mayor(int a, int b)

{

   if(a==b)

   printf("Son iguales\n\n");

   else

   {

        if(a>b)

printf("El valor de a es mayor que el de b\n\n");

        else

          printf("El valor de b es mayor que el de a\n\n");

   }

}
```

Definición de la Función

La función ha sido declarada, ha sido llamada y por lo tanto deber haber sido definida. Lo cual consta de dos partes, las cuales son:

1.La Primera Línea

Que como su nombre lo indica, es la primera línea de la definición de la función y con ella le indicamos al compilador que está en presencia de una función. Su formato es el siguiente:

Tipo_de_dato nombre_de_la_función (tipo y nombre de los argumentos)

2. Cuerpo de la función

Se inicia con una llave "{", y en ella, se pueden realizar asignaciones, cálculos, impresiones, así como la declaración de las variables locales. Puede estar constituidas por estructuras secuenciales, selectivas, iterativas, anidamientos, se pueden llamar otras funciones, etc; finaliza con "}". Puede devolver uno o ningún valor.

Ejemplo

Diseñe un Programa en C, que Dado un número entero y mayor que cero, Determine si es o no un número Primo. Ojo, los números primos sólo son divisibles por el mismo y por la unidad (1).

```
#include <stdio.h>

#include <conio.h>

void primo (int numero);

main()

 {

    int numero, ban=1;

    clrscr();

    while(ban==1)

{

    printf("Introduzca el n£mero por favor:\n");

scanf("%d", &numero);

    while(numero<0)

{

          printf("ERROR, el valor del n£mero debe ser mayor que cero\n");

           scanf("%d", &numero);

     }

    primo(numero);

   printf("¨Otro n£mero (si=1 y No=0)?\n");

scanf("%d", &ban);
```

```
    }
    getch();
    return 0;
}
 void primo (int numero)
 {
   int div, primo=1;
   for(div=2; div<numero; div++)
   {
     if(numero%div==0)
     {
          primo=0;
          printf("%d NO es primo\n\n\n", numero);
return 0;
     }
     else
          primo=1;
   }
   if(primo!=0)
printf("%d es primo\n\n\n", numero);
 }
```

5.5.1. Funciones que devuelven un valor entero

Las funciones que devuelven algún valor, se les llama PROTOTIPOS DE FUNCIONES:

Antes de usar una función C debe tener conocimiento acerca del tipo de dato que regresara y el tipo de los parámetros que la función espera.

El estándar ANSI de C introdujo una nueva (mejor) forma de hacer lo anterior respecto a las versiones previas de C.

La importancia de usar prototipos de funciones es la siguiente:

Se hace el código más estructurado y por lo tanto, más fácil de leer.

Se permite al compilador de C revisar la sintaxis de las funciones llamadas.

Lo anterior es hecho, dependiendo del alcance de la función. Básicamente si una función ha sido definida antes de que sea usada (o llamada), entonces se puede usar la función sin problemas.

Si no es así, entonces la función se debe declarar. La declaración simplemente maneja el tipo de dato que la función regresa y el tipo de parámetros usados por la función.

Es una práctica usual y conveniente escribir el prototipo de todas las funciones al principio del programa, sin embargo esto no es estrictamente necesario.

Para declarar un prototipo de una función se indicara el tipo de dato que regresará la función, el nombre de la función y entre paréntesis la lista del tipo de los parámetros de acuerdo al orden que aparecen en la definición de la función. Por ejemplo:

int longcad(int n); Lo anterior declara una función llamada longcad que regresa un valor entero y acepta otro valor entero como parámetro.

(Tomado de “Manual de C” de Héctor Tejada Villela)

Ejemplo 5.3

Diseñe un programa, que dado un número entero y mayor que cero, muestre su factorial. (El factorial de 5 es 120; 5x4x3x2x1=120)

```
#include <stdio.h>

#include <conio.h>

int factorial (int num);

main()

  {

     int num, ban=1;

     clrscr();

     while(ban==1)

     {
```

```
    printf("Ingrese el valor del n£mero por favor:\n");

    scanf("%d",& num);

    while(num<0)

                                        {

                printf("ERROR, el valor del n£mero debe ser mayor que cero:\n");

                                        scanf("%d",& num);

                                        }

    printf("El valor del factorial es %d\n\n", factorial (num));

    printf("¨Desea Realizar otro calculo?Si=1 y No=0\n");

    scanf("%d",& ban);

    }

  getch();

  return 0;

}

int factorial (int num)

{

 int sum=1, i;

 for(i=2; i<=num; i++)

  {

    sum=sum*i;

  }

 return (sum);

}
```

Explicación:

Quizá, lo único nuevo, e importante de explicar, radica en la llamada y la definición de la función. Cuando una función nos devolverá un valor entero, al identificador de dicha función debe

precederle el tipo de dato. En el lugar, donde llamamos la función, es que aparecerá el valor que nos devuelva, como valor de retorno. En nuestro ejemplo, en una impresión. Y al momento de definirla, no se nos debe olvidar, colocarle la sentencia *return();* ya que, mediante esta declaratoria, está retornando el valor calculado.

Pero, que sucede cuando se está trabajando, con valores bastante grandes, al utilizar solamente el int, se producirá un error lógico; ya que como valor de retorno podría ser un cero o una cifra negativa. Por tanto debemos usar el tipo de dato “long int”.

Ejemplo:

Diseñe un programa, que dada una cifra entera y mayor que cero, sea elevada a una potencia introducida por el usuario, la cual. (Ejemplo: 5^2=25).

```
#include <stdio.h>

#include <conio.h>

long int potencia (int base, int exponente);

main()

 {

   int base, exponente;

    clrscr();

    printf("La Base es:\n");

    scanf("%d", &base);

    while (base<0)

            {       printf("ERROR, el dato debe ser mayor que cero:\n");

                        scanf("%d", &base);

                }

    printf("El Exponente es:\n");

    scanf("%d", &exponente);

    printf("%d ^ %d es %ld\n\n", base, exponente, potencia(base,exponente));

    getch();

    return 0;

 }
```

```
long int potencia (int base, int exponente)

{

   long int sum=0, i,x;

   for(i=1; i<exponente; i++)

         {

                  x=base*base;

                    sum=sum+x;

                                                  }

   return (sum);

}
```

Este método es un poco complejo y puede realizarse de manera más fácil, haciendo uso de las funciones predefinidas en C, de las cuales hablaremos a continuación.

5.5.2. **Funciones que Devuelven un Valor Real**

Antes que nada, trataremos las funciones predefinidas en C. Ya que C, posee ciertas funciones que nos ayudan hacer nuestros programas más fáciles y utilizar menos código.

El lenguaje c, cuenta con una serie de funciones de bibliotecas que realizan operaciones y cálculos de uso frecuente.

Para acceder a una función, se realiza mediante el nombre seguido de los argumentos que le servirán a la función a realizar la tarea específica.

Nombre(arg1, arg2,...argn);

Nombre(arg1, arg2,...argn);

***Funciones Matemáticas**

Para acceder a ellas, se debe colocar la directiva #include <math.h> en el encabezado del programa.

Función (Sintaxis)	**Tipo de Dato**	**Propósito**
acos(d)	double	Devuelve el arco coseno de d
asin(d)	double	Devuelve el arco seno de d
atan(d)	double	Devuelve el arco tangente de d
atan(d1, d2)	double	Devuelve el arco tangente de d1/d2
ceil(d)	double	Devuelve el valor redondeado por exceso, al

		siguiente entero mayor
cos(d)	double	Devuelve el coseno de d
cosh(d)	double	Devuelve coseno hiperbólico de d
exp(d)	double	Eleva a la potencia d
fabs(d)	double	Devuelve el valor absoluto de d
floor(d)	double	Devuelve el valor redondeado por defecto al entero menor más cercano
log(d)	double	Devuelve el logaritmo natural de d
log10(d)	double	Devuelve el lo. (base10) de d
pow(d1, d2)	double	Devuelve d1 elevado a la potencia d2
sin(d)	Double	Devuelve el seno de d
sinh(d)	double	Seno hiperbólico de d
sqrt(d)	double	Raíz cuadrada de d
Tan(d)	double	Devuelve la tangente de d
tanh(d)	double	Devuelve la tangente hiperbólica de d

Las siguientes funciones se encuentran en las librerías: stdid.h ó stdlib.h:

Función (sintaxis)	**Tipo**	**Propósito**
abs(i)	int	Devuelve el valor absoluto de i
ran()	int	Devuelve un entero aleatorio
srand(u)	void	Inicializa el generador de números aleatorios
div(d1/d2)	Double/ int	Devuelve el cociente y el resto de la división
atuf(s)	Double	Convierte la cadena a una cantidad de doble precisión
atoi(s)	int	Convierte cadenas a un entero
atol(s)	long	Convierte cadenas a un entero largo

Hay muchas otras funciones, pero para ahondar más, debes saber cuál es la versión de C, instalada en tu máquina y así verificar cuáles funcionan correctamente; pero por lo general, estas funciones son muy estándar para la mayoría de compiladores.

A continuación, pasaremos a desarrollar una serie de ejercicios, en los cuales haremos uso de la funciones predefinidas en c, así como la modularización, es decir; el uso de funciones definidas por el usuario.

Ejemplo 5.5

Se desea conocer el resultado de las siguientes operaciones:

1. Ö a+b
2. |a-b|

Las variables a y b, son de tipo real, y pueden ser positivas o negativas.

```
#include <stdio.h>

#include <conio.h>

#include <math.h>

double raiz(float a, float b);

double valor_absoluto(float a, float b);

double exponente (float a, float b);

main()

{

float a, b;

clrscr();

printf("\t\tBIENVENIDO\n\n");

printf("Ingrese el valor de a, por favor:\n");

scanf("%f", &a);

printf("Ahora el valor de b:\n");

scanf("%f", &b);

printf("El resultado de la ra¡z cuadrada de %.2f + %.2f es %.2f\n\n", a,b,raiz(a,b));

printf("|%.2f-%.2f| es igual a %.2f\n\n", a,b,valor_absoluto(a,b));

printf("%.2f^%.2f es igual a %f\n\n", a,b,exponente(a,b));

getch();

return 0;

}

double raiz(float a, float b)

{

float x;

double y;

x=a+b;
```

```
y=sqrt(x);

return (y);

}

double valor_absoluto(float a, float b)

{

float x;

double y;

x=a-b;

y=fabs(x);

return (y);

}

double exponente (float a, float b)

{

double x;

x=pow(a,b);

return (x);

}
```

Supongo que, este ejemplo no requiere mayor explicación. Pero me gustaría que el lector, comprenda la gran cantidad de usos que podemos darle, a aquellas funciones matemáticas, junto con las funciones definidas por el usuario, esta es una gran ayuda, ya que ¿se imaginan la cantidad de código que deberíamos colocar, para determinar cosas tan elementales como el valor absoluto?; con estas funciones matemáticas, C, nos ahorra mucho trabajo y código

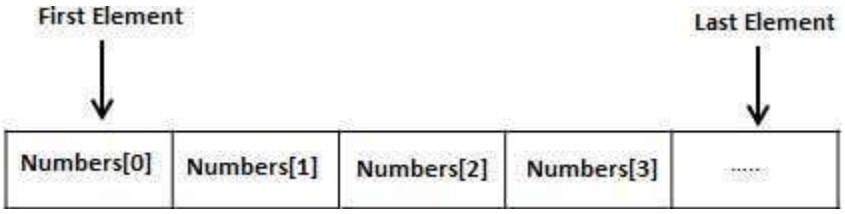
Capítulo No 6: Arreglos

6.1. Definición de Arreglos

Lenguaje de programación C proporciona una estructura de datos llamada la matriz, que puede almacenar un tamaño fijo de recogida secuencial de elementos del mismo tipo. Una matriz se utiliza para almacenar una colección de datos, pero a menudo es más útil pensar en una matriz como una colección de variables del mismo tipo.

En lugar de declarar las variables individuales, como Number0, número1, ..., y number99, se declara una variable de matriz tales como números y números de uso [0], números [1], y, números ... [99] para representar variables individuales. Un elemento específico de un array se accede por un índice.

Todos los arreglos consisten en posiciones de memoria contiguas. La dirección más baja corresponde al primer elemento y la dirección más alta al último elemento.



6.2. Declarar Arreglos (arrays)

Para declarar un ARREGLO en C, un programador especifica el tipo de los elementos y el número de elementos requeridos por una matriz como sigue:

type arrayName [arraySize];

Esto se llama una matriz unidimensional. El arraySize debe ser un entero constante mayor que cero y el tipo puede ser cualquier tipo de datos C válida. Por ejemplo, para declarar una matriz de 10 elementos llamado equilibrio de tipo double, utilice esta declaración:

double balance[10];

Ahora balance es matriz avariable que es suficiente para contener hasta 10 números dobles .:

6.3. La inicialización de arrays

Puede inicializar matriz en C, ya sea uno por uno o usando una sola declaración de la siguiente manera

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

El número de valores entre llaves {} no puede ser mayor que el número de elementos que declaramos para la matriz entre corchetes [].

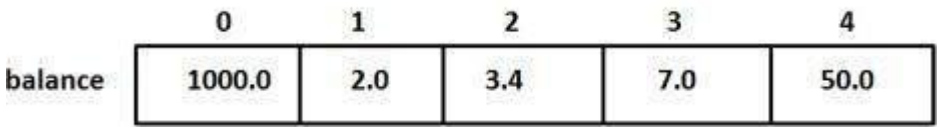
Si omite el tamaño de la matriz, se crea una matriz lo suficientemente grande para contener la inicialización. Por lo tanto, si usted escribe:

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

Va a crear exactamente la misma matriz como lo hizo en el ejemplo anterior. A continuación se presenta un ejemplo para asignar un único elemento de la matriz:

```
balance[4] = 50.0;
```

La declaración anterior asigna número de elemento quinto en la matriz con un valor de 50,0. Todas las matrices tienen 0 como el índice de su primer elemento que también se llama índice de base y último índice de un array será el tamaño total de la matriz menos 1. A continuación se presenta la representación gráfica de la misma matriz ya comentamos anteriormente



6.4. Acceso a elementos de un array

Un elemento que se accede por la indexación del nombre de la matriz. Esto se hace colocando el índice del elemento entre corchetes después del nombre de la matriz. Por ejemplo:

```
double salary = balance[9];
```

La declaración anterior se llevará a décimo elemento de la matriz y asignar el valor a la variable salario. A continuación se presenta un ejemplo que utilizará todos los mencionados tres conceptos anteriores a saber. declaración, misiones y con el acceso matrices:

```
#include <stdio.h>


int main ()

{

   int n[ 10 ]; /* n is an array of 10 integers */

   int i,j;
```

```
   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ )
   {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
}

   return 0;
}
```

Cuando se compila el código anterior y ejecutado, se produce el siguiente resultado:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

6.5. Arreglos en detalle

Las matrices son importantes para C y deben necesitar mucha más detalles. Hay siguiendo algunos conceptos importantes relacionados con la matriz que debe ser claro para un programador C:

Concepto	Descripción
Arreglos Multi-dimensional es	C admite matrices multidimensionales. La forma más simple de la matriz multidimensional es la matriz de dos dimensiones.
Pasando matrices a funciones	Puede pasar a la función un puntero a una matriz especificando el nombre de la matriz sin un índice.
Volver matriz desde una función	C permite una función para devolver una matriz.
Puntero a una matriz	Puede generar un puntero al primer elemento de una matriz simplemente especificando el nombre de la matriz, sin ningún índice

6.6. Arreglos unidimensionales

El siguiente ejemplo muestra la definición de tres arreglos, uno de 80 elementos doble precisión, otrode 30 elementos enteros y uno de 20 elementos tipo carácter.

```
double x[80];

int factores[30];

char codSexo[20];
```

Los nombres deben cumplir con las normas para los identificadores. La primera línea indica que se hanreservado 80 posiciones para números doble precisión. Estas posiciones son contiguas. Es importanterecalcar que en C, a diferencia de otros lenguajes, el primer elemento es x[0], el segundo es x[1], eltercero es x[2], y así sucesivamente; el último elemento es x[79].

En x hay espacio reservado para 80 elementos, pero esto no obliga a trabajar con los 80; el programapuede utilizar menos de 80 elementos.C no controla si los subíndices están fuera del rango previsto; esto es responsabilidad del programador.

Por ejemplo, si en algún momento el programa debe utilizar x[90], lo usa sin importar si los resultadosson catastróficos.

Cuando un parámetro de una función es un arreglo, se considera implícitamente que es un parámetropor referencia. O sea, si en la función se modifica algún elemento del arreglo, entonces se modificó realmenteel valor original y no una copia. Pasar un arreglo como parámetro de una función y llamar estafunción es muy sencillo. Se hace como en el esquema siguiente

```
... funcion(..., double x[], ...); // prototipo
```

```
//-------------------------------------------------

int main(void)

{

double v[30];

...

... funcion(..., v, ...); // llamado a la funcion

...

}

//-------------------------------------------------

... funcion(..., double x[],...)// definicion de la funcion

{

// cuerpo de la funcion

...

}
```

En el esquema anterior, el llamado a la función se hizo desde la función main. Esto no es ningunaobligación; el llamado se puede hacer desde cualquier función donde se define un arreglo o donde a suvez llega un arreglo como parámetro.También se puede hacer el paso de un arreglo como parámetro de la siguiente manera. Es la formamás usual

```
... funcion(..., double *x, ...); // prototipo

//-------------------------------------------------

int main(void)

{

double v[30];

...

... funcion(..., v, ...); // llamado a la funcion

...
```

```
}

//-------------------------------------------------

... funcion(..., double *x, ...)// definicion de la funcion

{

// cuerpo de la funcion

...

}
```

El programa del siguiente ejemplo lee el tamaño de un vector, lee los elementos del vector, los escribe yhalla el promedio. Para esto utiliza funciones. Observe la manera como un arreglo se pasa como parámetro.

```
// Arreglos unidimensionales

// Lectura y escritura de un vector y cálculo del promedio

//-------------------------------------------------

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

//-------------------------------------------------

void lectX(double *x, int n, char c );

void escrX(double *x, int n );

double promX( double *x, int n);

//================================================

int main()

{

double v[40];

int n;

printf("\n Promedio de elementos de un vector.\n\n");
```

6.7. ARREGLOS UNIDIMENSIONALES

```
printf(" numero de elementos : ");

scanf( "%d", &n);

if( n > 40 ){

printf("\n Numero demasiado grande\n\n");

exit(1);

}

lectX(v, n, 'v');

printf(" v : \n");

escrX(v, n);

printf(" promedio = %lf\n", promX(v, n));

return 0;

}

//================================================

void lectX(double *x, int n, char c )

{

// lectura de los elementos de un "vector".

int i;

for( i = 0; i < n; i++){

printf(" %c(%d) = ", c, i+1);

scanf("%lf", &x[i] );

}

}

//------------------------------------------------

void escrX(double *x, int n )
```

```
{

// escritura de los elementos de un "vector".

int i;

int nEltosLin = 5; // numero de elementos por linea

for( i = 0; i < n; i++){

printf("%15.8lf", x[i]);

if( (i+1)%nEltosLin == 0 || i == n-1) printf("\n");

}

}

//-------------------------------------------------

double promX( double *x, int n)

{

// promedio de los elementos del 'vector' x

int i;

double s = 0.0;

if( n <= 0 ){

printf(" promX: n = %d inadecuado\n", n);

return 0.0;

}

for( i = 0; i < n; i++) s += x[i];

return s/n;

}
```

La función lectX tiene tres parámetros: el arreglo, el número de elementos y una letra. Esta letra sirvepara el pequeño aviso que sale antes de la lectura de cada elemento. En el ejemplo, cuando se "llama" lafunción, el tercer parámetro es 'v'; entonces en la ejecución aparecerán los avisos:

v(1) =

v(2) =

...

Observe que en el printf de la función lectX aparece i+1; entonces para el usuario el “vector” empiezaen 1 y acaba en n. Internamente empieza en 0 y acaba en n − 1.

Cuando un arreglo unidimensional es parámetro de una función, no importa que el arreglo haya sidodeclarado de 1000 elementos y se trabaje con 20 o que haya sido declarado de 10 y se trabaje con 10. Lafunción es de uso general siempre y cuando se controle que no va a ser llamada para usarla con subíndices mayores que los previstos. En la siguiente sección se trata el tema de los arreglos bidimensionales. Allí,el paso de parámetros no permite que la función sea completamente general

En el siguiente ejemplo, dado un entero n ≥ 2 (pero no demasiado grande), el programa imprime losfactores primos. El algoritmo es muy sencillo. Se busca d > 1, el divisor más pequeño de n. Este divisores necesariamente un primo. Se divide n por d y se continúa el proceso con el último cociente. El procesotermina cuando el cociente es 1. Si n = 45, el primer divisor es 3. El cociente es 15. El primer divisor de15 es 3. El cociente es 5. El primer divisor de 5 es 5 y el cociente es 1.// Arreglos unidimensionales

```
// Factores primos de un entero >= 2

//-------------------------------------------------

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

//-------------------------------------------------

int primerDiv( int n);

int factoresPrimos( int n, int *fp, int &nf, int nfMax);

//================================================

int main()

{

int vFactPrim[40]; // vector con los factores primos

int n;

int nFact; // numero de factores primos
```

```
int i;

printf("\n Factores primos de un entero >= 2.\n\n");

printf(" n = ");

scanf( "%d", &n);

if( factoresPrimos(n, vFactPrim, nFact, 40) ){

for(i = 0; i < nFact; i++) printf(" %d",

vFactPrim[i]);

printf("\n");

}

else printf(" ERROR\n");

return 0;

}

//================================================

int primerDiv( int n)

{

// n debe ser mayor o igual a 2.

// Calcula el primer divisor, mayor que 1, de n

// Si n es primo, devuelve n.

// Si hay error, devuelve 0.

int i;

if( n < 2 ){

printf(" primerDiv: %d inadecuado.\n", n);

return 0;

}

for( i = 2; i*i <= n; i++) if( n%i == 0 ) return i;
```

```
return n;

}

//================================================

int factoresPrimos( int n, int *fp, int &nf, int nfMax)

{

// factores primos de n

// devuelve 0 si hay error.

// devuelve 1 si todo esta bien.

// fp : vector con los factores primos

// nf : numero de factores primos

// nfMax : tamano del vector fp

int d, indic;

if( n < 2 ){

printf(" factoresPrimos: %d inadecuado.\n", n);

return 0;

}

nf = 0;

do{

if( nf >= nfMax ){

printf("factoresPrimos: demasiados factores.\n");

return 0;

}

d = primerDiv(n);

fp[nf] = d;

nf++;
```

```
n /= d;

} while( n > 1);

return 1;

}
```

6.8. Arreglos multidimensionales:

La declaración de los arreglos bidimensionales, caso particular de los arreglos multidimensionales, sehace como en el siguiente ejemplo:

```
double a[3][4];

int pos[10][40];

char list[25][25];
```

En la primera línea se reserva espacio para 3 × 4 = 12 elementos doble precisión. El primer subíndice varıa entre 0 y 2, y el segundo varıa entre 0 y 3. Usualmente, de manera análoga a las matrices, se diceque el primer subındice indica la fila y el segundo subındice indica la columna.

Un arreglo tridimensional se declararía así:

```
double c[20][30][10];
```

Los sitios para los elementos de a están contiguos en el orden fila por fila, o sea, a[0][0], a[0][1],

a[0][2], a[0][3], a[1][0], a[1][1], a[1][2], a[1][3], a[2][0], a[2][1], a[2][2], a[2][3].

En el siguiente ejemplo, el programa sirve para leer matrices, escribirlas y calcular el producto. Lohace mediante la utilización de funciones que tienen como parámetros arreglos bidimensionales.

```
// Arreglos bidimensionales

// Lectura y escritura de 2 matrices y cálculo del producto

//-------------------------------------------------

#include <math.h>

#include <stdio.h>

#include <stdlib.h>
```

```
//-------------------------------------------------

void lectA0(double a[][40], int m, int n, char c );

void escrA0(double a[][40], int m, int n );

int prodAB0(double a[][40], int m, int n, double b[][40],

int p, int q, double c[][40]);

//=================================================

int main()

{

double a[50][40], b[20][40], c[60][40];

int m, n, p, q;

printf("\n Producto de dos matrices.\n\n");

printf(" num. de filas de A : ");

scanf( "%d", &m);

printf(" num. de columnas de A : ");

scanf( "%d", &n);

// es necesario controlar que m, n no son muy grandes

// ni negativos

printf(" num. de filas de B : ");

scanf( "%d", &p);

printf(" num. de columnas de B : ");

scanf( "%d", &q);

// es necesario controlar que p, q no son muy grandes


// ni negativos

if( n != p ){
```

```
printf(" Producto imposible\n");

exit(1);

}

lectA0(a, m, n, 'A');

printf(" A : \n");

escrA0(a, m, n);

lectA0(b, n, q, 'B');

printf(" B : \n");

escrA0(b, n, q);

if( prodAB0(a,m,n, b,p,q, c) ){

printf(" C : \n");

escrA0(c, m, q);

}

else printf("\ ERROR\n");

return 0;

}

//================================================

void lectA0(double a[][40], int m, int n, char c )

{

// lectura de los elementos de una matriz.

int i, j;

for( i = 0; i < m; i++){

for( j=0; j < n; j++){

printf(" %c[%d][%d] = ", c, i+1, j+1);

scanf("%lf", &a[i][j] );
```

```
}

}

}


//-----------------------------------------------

void escrA0(double a[][40], int m, int n )

{

// escritura de los elementos de una matriz

int i, j;

int nEltosLin = 5; // numero de elementos por linea

for( i = 0; i < m; i++){

for( j = 0; j < n; j++){

printf("%15.8lf", a[i][j]);

if((j+1)%nEltosLin == 0 || j==n-1)printf("\n");

}

}

}

//-----------------------------------------------

int prodAB0(double a[][40], int m, int n, double b[][40],

int p, int q, double c[][40])

{

// producto de dos matrices, a mxn, b pxq

// devuelve 1 si se puede hacer el producto

// devuelve 0 si no se puede

int i, j, k;
```

```
double s;

if(m<0||n<0||p<0||q<0 || n!= p ) return 0;

for( i=0; i < m; i++){

for( j=0; j < q; j++){

s = 0.0;

for( k=0; k<n; k++) s += a[i][k]*b[k][j];

c[i][j] = s;

}

}

return 1;

}
```

6.9. Cadenas de caracteres

En C no existe un tipo predefinido para manipular cadenas de caracteres (string). Sinembargo, el estándar de C define algunas funciones de biblioteca para tratamiento decadenas.

Una cadena en C es un array de caracteres de una dimensión (vector de caracteres) quetermina con el carácter especial '\0' (cero).

El formato para declarar una cadena es:

```
char nombre[n];
```

donde: n >= 1 y representa a la longitud-1 real de la cadena.

Un ejemplo de declaración de cadena:

```
char cadena [5];
```

Debido a que en la representación interna de una cadena de caracteres es terminada porel símbolo '\0', para un texto de "n" caracteres, debemos reservar "n+1". El carácter '\0',aunque pertenece a la cadena, no aparece al utilizar funciones como printf.

En el caso especial de los arrays de caracteres, podemos utilizar varias formas deinicialización:

```
char cadena[] = "Hola";

char cadena[] = {'H','o','l','a',0};
```

char cadena[] = {'H','o','l','a','\0'};

sin especificar el tamaño de la cadena, o especificando el tamaño:

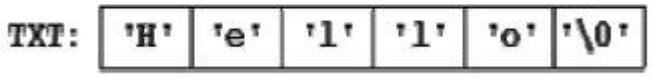
char cadena[5] = "Hola";

char cadena[5] = {'H','o','l','a',0};

char cadena[5] = {'H','o','l','a','\0'};

Durante la inicialización, se reserva automáticamente el número de bytes necesariospara la cadena, esto es, el número de caracteres más uno. Por ejemplo:

```
char TXT[] = "Hello";
```



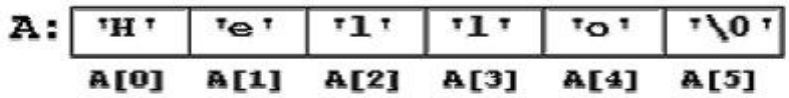
Para acceder a un elemento de una cadena de caracteres puede hacerse de la mismamanera que el acceso al elemento de un array.

cadena[i];

donde: 0 <=i < n

Por ejemplo:

```
char A[6] = "Hello";
```



La biblioteca “string” tiene una gran cantidad de funciones prácticas para trabajar con

cadenas de caracteres. Para utilizarlas debemos de incluir el fichero que define los

prototipos de dichas funciones:

#include <string.h>

Algunas de las funciones más importantes son:

• strlen(<cadena>): Devuelve la longitud de la cadena sin tomar en cuentael carácter de final de cadena.

• strcpy(<cadena_destino>, <cadena_origen>) : Copia elcontenido de <cadena_origen> en <cadena_destino>.

• strcat(<cadena_destino>, <cadena_origen>) : Concatena elcontenido de <cadena_origen> al final de <cadena_destino>.

• strcmp(<cadena1>, <cadena2>) : Compara las dos cadenas ydevuelve un 0 si las dos cadenas son iguales, un número negativo si <cadena1> es menor que (precede alfabéticamente a) <cadena2> y unnúmero positivo (mayor que cero) si <cadena1> es mayor que <cadena2>.

A diferencia de los arrays de tipos de datos numéricos (arrays de enteros, de númeroscon punto decimal, etc.), en donde cada elemento del array se debe considerar como unavariable independiente de los demás, los arrays de caracteres (cadenas) se puedenmanipular de dos maneras: de forma conjunta o separada.

Por ejemplo, para mostrar en pantalla un array de caracteres podemos hacerlo dentro deun bucle, desde el primer caracter (indice 0) hasta el último carácter (lo que nos

devuelve la función strlen):

```
for(i=0; i<strlen(cadena); i++)

printf("%c",cadena[i]);
```

Existe una mejor manera de mostrar en pantalla una cadena, y es utilizando el carácterde conversión %s:

```
printf("%s",cadena);
```

El tipo Carácter en C El nombre del tipo carácter en C es char. Este tipo ocupa un byte (ocho bits) de extensión, y puede interpretarse con y sin signo. La tabla de tipos de datos de carácter en C es como sigue:

Nombre	Extensión	Alcance
`char`	8 bits	Cualquier código ASCII
`signed char`	8 bits	desde -128 hasta +127
`unsigned char`	8 bits	desde 0 hasta 255

Véase a continuación un ejemplo de utilización de variables de los tipos anteriores:

```
#include<stdio.h>



char letra;
```

```
signed char letraconsigno;

unsigned char letrasinsigno;


int main(int argc, char * argv[])

{

    letra = 'A';

    printf("La letra es %c y su valor decimal es %d.\n\n",

                                letra,letra);


    letraconsigno = -65;

    printf("Letraconsigno es %d y su valor decimal es %d.\n\n",

                                letraconsigno,letraconsigno);


    letrasinsigno = 165;

    printf("Letrasinsigno es %u y su valor decimal es %d.\n\n",

                                letrasinsigno,letrasinsigno);

    return 0;

}
```

Y el resultado de ejecutar este programa es:

La letra es A y su valor decimal es 65.

Letraconsigno es -65 y su valor decimal es -65.

Letrasinsigno es 165 y su valor decimal es 165.

Ejemplo

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

```
/* Este programa muestra la utilización de cadenas en C */


char cadena[8], resto[80];


void main(void)

{

        printf("Escriba una cadena: ");

        scanf("%s", cadena); /* Las cadenas NO llevan & */


        printf("\n\nLa cadena leída era %s\n\n", cadena);


        printf("Y el resto era");

        gets(resto);

        puts(resto);

        printf("Terminación normal del programa.\n\n");

}
```

Es interesante recordar que todo lo que "sobra", esto es, lo que está más allá del primer espacio en blanco, queda almacenado y disponible para la siguiente sentencia de lectura. La función gets() captura todo lo restante, y después se imprime mediante puts(). Hay que ser precavidos a la hora de "dejar" restos en el tampón. Lo mejor es eliminar posibles restos antes de pasar a la lectura siguiente, empleandofflush(stdin) o bien fpurge(stdin).

Bibliografía

Quereda, J. M. (2000). *Introducción a la programación con Pascal*. Universitat Jaume.
WordReference. (11 de 01 de 2015). *Word Reference*. Obtenido de Word Reference: http://www.wordreference.com/definicion/conceptualizaci%C3%B3n

ALGORITMOS RESUELTOS CON DIAGRAMASDE FLUJO Y PSEUDOCÓDIGO, Francisco Javier Pinales Delgado, César Eduardo Velázquez Amador (2013)

http://www.monografias.com/trabajos15/algoritmos/algoritmos.shtml

http://es.wikipedia.org/wiki/Algoritmo

http://www.areatecnologia.com/informatica/ejemplos-de-diagramas-de-flujo.html

http://www.uhu.es/04004/material/El_Entorno_DevC.pdf