

MANUAL ASIGNATURA INTRODUCCIÓN A LA PROGRAMACIÓN

**VERSIÓN 7.3
Junio 2009**

Área Informática & Telecomunicaciones

Colaboraron en el presente manual:

Versión 5.3

Enrique Castillo
Jefe de Carrera sede Osorno

Jorge Douglas
Asesor de la DAI – MCP

Luis Flores
Docente sede de Ñuñoa – CCNA

Luis Aguirre Tapia
Jefe de Carrera Sede Copiapó

Carlos Dides
Asesor de la DAI – CCNA

Lidia Herrera Mateluna
Jefe de Carrera sede de Viña del Mar

Versión 5.4

Adrian Anex M.
Asesor de la DAI

Leonardo Bolton Montalva
Director de Área

Versión 5.5

Ernesto Ramos V.
Docente Sede La Serena

Versión 6.0

Miguel Ortega
Docente Sede Ñuñoa

Versión 7.3

María E. Sepúlveda Berrocal
Docente Sede Santiago Sur

Derechos Reservados

Titular del derecho: INACAP

Nº de inscripción en el Registro de Propiedad Intelectual #..... de fecha

© INACAP 2009.

i. Introducción

El objetivo de la asignatura “Introducción a la Programación” es introducir en forma progresiva y sistemática una correcta metodología para que los alumnos de la Carrera Ingeniería en Informática dominen las técnicas de la programación de computadores.

Este manual esta confeccionado con el propósito de servir de **guía o ayuda** para el desarrollo de la asignatura. Las materias aquí tratadas corresponden a lo indicado en el Programa Oficial de Inacap por lo que deberán ser tratadas en clases en su totalidad y siguiendo los lineamientos que aquí se indican.

Se debe tener en cuenta que la Asignatura “Introducción a la Programación” está orientada esencialmente a la metodología, por lo tanto, se persigue que el alumno adquiera los conocimientos necesarios para desarrollar programas utilizando métodos precisos, claros y eficientes.

La orientación de las técnicas de solución de problemas **no debe** estar orientada hacia ningún lenguaje de programación en especial, hasta la unidad 2. Desde la unidad 3 en adelante, la sintaxis a usar para el desarrollo de algoritmos debe ser la correspondiente al lenguaje C.

ii. Estructura del manual

El manual está dividido en unidades. Al final de cada una de ellas se dejan algunos “Ejercicios propuestos” para entregárselos a los alumnos o desarrollarlos en clases. Todos los ejercicios están solucionados al final del manual. (ANEXO 2).

iii. Nomenclatura

Con el objeto de estandarizar la nomenclatura utilizada en la resolución de problemas mediante un **pseudolenguaje único a nivel nacional**, se han adoptado las siguientes normas:

- Todas las palabras claves de los algoritmos son en ingles. Ejemplo: **While , If, End, Read, Print, Open, Close, Select**, etc
- Todas las variables deben ser declaradas. Se utilizan solamente dos tipos de variables: numéricas y alfanuméricas. El prefijo para declarar las variables numéricas es **num** y para las variables alfanuméricas, **char**. Se debe anteponer al nombre de la variable

Ejemplos: num x
 char frase

- La asignación de variable utiliza el símbolo **:=**
- El signo igual = sólo se debe utilizar como operador relacional de igualdad.
- El índice menor de un arreglo es 0.
- Los comentarios se identifican anteponiendo el símbolo #

iv. Ejemplos y ejercicios

Se ha incorporado una gran cantidad de ejemplos de algoritmos. Al final de cada unidad se han propuesto una serie de ejercicios cuya solución se ha desarrollado al final del manual.

Dado que el proceso de revisión de esos ejercicios no alcanzó a ser optimizado, se hace **indispensable** que los docentes revisen la sintaxis y la lógica de los algoritmos antes de utilizarlos como material didáctico. Los errores descubiertos deberían comunicarse a la DAI para la modificación del manual.

En relación a los programas desarrollados en C, es necesario considerar que no están incluidas las instrucciones que permiten generar pausas para poder visualizar los resultados correctamente. Además se realizaron en Dev C++.

ÍNDICE DE CONTENIDOS

UNIDAD 1: ANÁLISIS DE PROBLEMAS Y PROCESAMIENTO DE DATOS

OBJETIVO 1.1: RESOLVER PROBLEMAS GENERALES APLICANDO LA METODOLOGÍA DE POLYA

- 1.1.1. Las Estrategias de Resolución de problemas.....Pág.8
- 1.1.2. Metodología de Resolución de Problemas Según Polya.....Pág.9
- 1.1.3. Algunas Sugerencias hechas por quienes tienen éxito en resolver problemas.....Pág.13

OBJETIVO 1.2: ANALIZAR UN PROCESO COMPUTACIONAL EN TÉRMINOS DE ENTRADA – OPERACIÓN INTERNA – SALIDA

- 1.2.1. Concepto de Dato e Información.....Pág.14

UNIDAD 2: ALGORITMOS Y ESTRUCTURAS DE CONTROL

OBJETIVO 2.1: DESCRIBIR EL PROCESO DE CREACIÓN DE UN PROGRAMA PARA PROBLEMAS SIMPLES

- 2.1.1. Procesos de Creación de Programas.....Pág.16

OBJETIVO 2.2: EXPLICAR LOS PRINCIPIOS QUE RIGEN EL ALMACENAMIENTO DE DATOS

- 2.2.1. Principios de Almacenamiento.....Pág.18

OBJETIVO 2.3: DESCRIBIR A TRAVÉS DE DEFINICIONES Y EJEMPLOS CADA UNO DE LOS ELEMENTOS QUE COMPONEN UN ALGORITMO

- 2.3.1. Concepto de Algoritmo.....Pág.22
- 2.3.2. Estructura de un Algoritmo.....Pág.22

OBJETIVO 2.4: REPRESENTAR GRÁFICAMENTE LA CIRCULACIÓN DE DATOS E INFORMACIÓN Y LA SECUENCIA DE OPERACIONES INVOLUCRADAS EN UN ALGORITMO DADO

- 2.4.1. Definición de Diagrama de Flujo.....Pág.24
- 2.4.2. Simbología de Diagrama de Flujo.....Pág.24

OBJETIVO 2.5: DESCRIBIR CADA UNO DE LOS ELEMENTOS QUE COMPONEN UN ALGORITMO EN PSEUDOLENGUAJE

- 2.5.1. Definición de Pseudolenguaje.....Pág.35
- 2.5.2. Tipos de Datos.....Pág.35
- 2.5.3. Variables.....Pág.35
- 2.5.4. Constantes.....Pág.37

2.5.5. Operadores.....	Pág.37
2.5.6. Tipo de Expresiones.....	Pág.43

OBJETIVO 2.6: DESARROLLAR ALGORITMOS QUE INCLUYEN MANEJO DE VARIABLES, ENTRADA, SALIDA Y ESTRUCTURAS DE CONTROL

2.6.1. Concepto de Instrucciones.....	Pág.46
2.6.2. Clasificación de las Instrucciones.....	Pág.46
2.6.3. Estructura de los algoritmos en pseudolenguaje.....	Pág.46
2.6.4. Nomenclatura de instrucciones que se utilizarán en pseudolenguaje.....	Pág.46
2.6.5. Estructuras de Selección.....	Pág.48
2.6.6. Instrucciones de Control de Repetición.....	Pág.56
2.6.7. Formas Comunes de Estructuras de repetición.....	Pág.57

UNIDAD 3: INTRODUCCIÓN AL LENGUAJE C

OBJETIVO 3.1: REVISAR LOS ELEMENTOS DE SINTAXIS Y ESTRUCTURA DE UN PROGRAMA EN LENGUAJE C.

3.1.1. Describe las etapas de un programa computacional en lenguaje C.....	Pág.68
3.1.2. Revisa la estructura de un programa en C.....	Pág.68
3.1.3. Explica el procedimiento de compilación de programas en lenguaje C.....	Pág.70

OBJETIVO 3.2: TRADUCIR LOS ALGORITMOS ESCRITOS EN PSEUDOLENGUAJE A LENGUAJE C, DETECTANDO Y CORRIGIENDO ERRORES PARA UNA EJECUCIÓN SATISFACTORIA.

3.2.1. Explica los tipos de datos soportados por el lenguaje C.....	Pág. 73
3.2.2. Reconoce los operadores aritméticos, de asignación y relacionales en lenguaje C....	Pág. 74
3.2.3. Reconoce la sintaxis de las instrucciones de Entrada y Salida en lenguaje C.....	Pág. 77
3.2.4. Reconoce la sintaxis de las instrucciones de decisión en lenguaje C.....	Pág. 79
3.2.5. Reconoce la sintaxis de las instrucciones de repetición en lenguaje C.....	Pág. 81
3.2.6. Codifica programas fuente en C, según su correspondiente pseudolenguaje.....	Pág. 85

UNIDAD 4: SUB PROCESOS

OBJETIVO 4.1: DESCRIBE EL PROCESO PROGRAMACIÓN MODULAR

4.1.1. Identifica la estructura general de un programa.....	Pág.90
4.1.2. Identifica las partes principales de un proceso modular.....	Pág.93
4.1.3. Reconoce la relación entre programación estructurada y prog. Modular.....	Pág.94

OBJETIVO 4.2: DESCOMPONE UN PROBLEMA GENERAL EN SUB-PROBLEMAS O MÓDULOS

4.2.1. Reconoce cuando es necesario modularizar un problema general.....	Pág.95
4.2.2. Descompone un problema general en subproblemas o módulos.....	Pág.96
4.2.3. Reconoce los tipos de sub-módulos y sus estructuras.....	Pág.97

OBJETIVO 4.3: RESUELVE PROBLEMAS HACIENDO USO DE SUB-PROGRAMAS A TRAVES DE PROCEDIMIENTOS Y FUNCIONES.

4.3.1. Establece la relación entre el modulo principal y los sub-módulos.....	Pág.99
4.3.2. Reconoce los tipos de variables en términos de su alcance.....	Pág.106
4.3.3. Analiza ventajas y desventajas de la modularización de un problema.....	Pág.108
4.3.4. Realiza programas en C, usando funciones.....	Pág.109

UNIDAD 5: REPRESENTAR ESTRUCTURAS DE DATOS

OBJETIVO 5.1: REPRESENTAR ESTRUCTURAS DE ARREGLOS DE ACUERDO A SUS COMPONENTES.

5.1.1. Estructuras de Datos elementales.....	Pág. 110
5.1.2. Concepto de Arreglo.....	Pág. 110
5.1.3. Arreglos Unidimensionales o Vectores.....	Pág. 111
5.1.4. Arreglos Bidimensionales.....	Pág. 112

OBJETIVO 5.2:CONSTRUYE ALGORITMOS Y PROGRAMAS EN C UTILIZANDO PROCESOS TÍPICOS CON ARREGLOS.

5.2.1. Declaración, ingreso y salida de datos.....	Pág. 113
5.2.2. Manipulación de Arreglos.....	Pág. 115
5.2.3. Procesos típicos sobre arreglos.....	Pág. 117
5.2.4. Ordenamiento y Búsqueda en Vectores.....	Pág. 125

ANEXO 1

GLOSARIO.....	Pág. 130
----------------------	-----------------

ANEXO 2

SOLUCION A EJERCICIOS PROPUESTOS.....	Pág. 134
--	-----------------

UNIDAD 1: ANALISIS DE PROBLEMAS Y PROCESAMIENTO DE DATOS

OBJETIVO 1.1: RESOLVER PROBLEMAS GENERALES APLICANDO LA METODOLOGÍA DE POLYA

Uno de los aspectos más importantes a considerar en el proceso de resolución de problemas es conseguir una representación mental del problema. Esto implica tener una visión general del problema que hace mucho más fácil entender sus características y singularidades. Los datos importantes son identificados y relacionados entre sí. Al modelado del problema se le llama también *espacio del problema*.

A continuación se presenta un ejemplo para ilustrar cómo normalmente se representa un problema en nuestra mente:

Un autobús parte del terminal en la mañana. Se detiene en la primera parada y recoge 5 personas. Sigue hasta la próxima parada y allí suben 6 personas. Continúa hasta la siguiente parada y suben 4 personas. En la próxima parada, suben 5 personas y se bajan 3. En la siguiente parada, suben 5 personas y se bajan 4. En la parada siguiente, suben 6 personas y se baja 1. La próxima vez, suben 3 personas y se bajan 2. La vez siguiente, se bajan 2 personas y no sube nadie. En la siguiente parada nadie espera por el autobús, de manera tal que este no se detiene. Finalmente, el autobús llega al terminal.

¿Cuántas paradas hay en la ruta del autobús?

La tendencia más común es calcular cuántas personas llegan a la parada final, cuántas subieron o cuántas bajaron, pero muy pocos están en condiciones de indicar cuántas paradas hay en la ruta del autobús debido a que seleccionaron la información numérica como dato importante y la representaron internamente en la forma de operaciones aritméticas.

En términos de los procesos involucrados en resolución de problemas, esto sucede porque la meta del problema no está bien definida a pesar de que hay datos numéricos explícitos precisos. El énfasis sobre el número de personas que suben y bajan del autobús hace posible que los estudiantes piensen que tienen que hacer algo con esos datos y, en tal sentido, construyen una meta la cual se representa como el logro de una cantidad total. Esta decisión conduce a los estudiantes a seleccionar cierta información como relevante (número de personas que suben y bajan del autobús) e ignorar otra (número de paradas del autobús).

1.1.1. Las estrategias de resolución de problemas

Las estrategias para resolver problemas se refieren a las operaciones mentales utilizadas para pensar sobre la representación de las metas y los datos, con el fin de transformarlos para obtener una solución. Las estrategias para la resolución de problemas incluyen los métodos heurísticos, los algoritmos y los procesos de pensamiento divergente.

1.1.1.a. Los métodos heurísticos

Son estrategias generales de resolución y reglas de decisión utilizadas por los solucionadores de problemas, basadas en la experiencia previa con problemas similares. Estas estrategias

indican las vías o posibles enfoques a seguir para alcanzar una solución pero, no garantiza la solución del problema.

1.1.1.b. Los algoritmos

Los algoritmos son procedimientos específicos que señalan paso a paso la solución de un problema y que garantizan el logro de una solución.

Un procedimiento algorítmico es una sucesión de acciones que hay que realizar, completamente prefijada. Su correcta ejecución lleva a una solución segura del problema como por ejemplo, realizar una raíz cuadrada, pegar un botón, preparar una taza de café, hacer sopaipillas, etc. El algoritmo garantiza la obtención de lo que nos proponemos.

De esta manera, el algoritmo se diferencia del heurístico en que este último constituye sólo “una buena apuesta”, ya que ofrece una probabilidad razonable de acercarnos a una solución. Por lo tanto, es aceptable que se utilicen los procedimientos heurísticos en vez de los algorítmicos cuando no conocemos la solución de un problema.

1.1.1.c. Los procesos de pensamiento divergente

Los procesos de pensamiento divergente permiten la generación de enfoques alternativos a la solución de un problema y están relacionados principalmente con la inspiración y la creatividad.

La adquisición de habilidades para resolver problemas ha sido considerada como el aprendizaje de sistemas de producción que involucran tanto el conocimiento declarativo como el procedimental. Existen diversos procedimientos que pueden facilitar o inhibir la adquisición de habilidades para resolver problemas, entre los cuales se pueden mencionar:

- ✓ Ofrecer representaciones metafóricas.
- ✓ Permitir la verbalización durante la solución del problema.
- ✓ Hacer preguntas.
- ✓ Ofrecer ejemplos.
- ✓ Ofrecer descripciones verbales.
- ✓ Trabajar en grupo.
- ✓ Utilizar auto-explicaciones.

1.1.2. METODOLOGÍA DE RESOLUCIÓN DE PROBLEMAS SEGÚN POLYA.

Existen métodos bien definidos y desarrollados para lograr una resolución efectiva de problemas. Uno de estos métodos es el desarrollado por George Polya destacado Doctor en Matemática, nacido en Hungría en 1887, el que fue publicado inicialmente en la Universidad de Princeton en 1945 y finalmente en su libro "How to Solve It" en 1957.

Las sugerencias de Polya para responder a la pregunta ¿Cómo plantear y resolver problemas?, son las siguientes:

- ✓ **Entender el problema.** (reconociendo qué se pregunta.)
- ✓ **Idear un plan.** (respondiendo a lo que se pide.)
- ✓ **Realizar el plan.** (desarrollando el resultado de la respuesta.)
- ✓ **Mirar hacia atrás.** (controlando qué hace y dice el resultado)

1.1.2.a Entender el problema. Consiste en identificar qué se pide de modo completamente independiente de las diversas condiciones que pueden ser impuestas y limitaciones constatables en el problema. Se deben considerar aspectos como:

- ✓ ¿Entiende todo lo que dice?
- ✓ ¿Puede replantear el problema en sus propias palabras?
- ✓ ¿Distingue cuáles son los datos?
- ✓ ¿Sabe a qué quiere llegar?
- ✓ ¿Hay suficiente información?
- ✓ ¿Hay información extraña?
- ✓ ¿Es este problema similar a algún otro que haya resuelto antes?

1.1.2.b Idear un plan. Consiste de responder tan directamente como sea posible lo que se pide. Esto requiere generalmente uso de una ley, de una definición o de un principio que sea la respuesta a la pregunta hecha. Encontrar la conexión entre los datos y lo que se quiere. Puede verse obligado a considerar problemas auxiliares si una conexión inmediata no puede ser encontrada. Se deben considerar aspectos como:

- ✓ ¿Ha visto el mismo problema en una forma levemente diferente?
- ✓ ¿Conoce algún problema relacionado?
- ✓ Si está frente a algo completamente desconocido, intente pensar en un problema conocido que tenga alguna parte igual o similar.
- ✓ Si es un problema relacionado con otro solucionado antes. ¿Podría utilizarlo? ¿Podría utilizar su resultado, su método?
- ✓ ¿Podría exponer el problema en forma modificada?
- ✓ ¿Se usaron todos los datos?

1.1.2.c Realizar el plan. Consiste en el responder a las peticiones hechas por el resultado del paso anterior. Esto puede requerir revisar nuevamente los pasos anteriores. Se deben considerar aspectos como:

- ✓ Implementar la o las estrategias que se escogieron hasta solucionar completamente el problema o hasta que la misma acción sugiera tomar un nuevo curso.
- ✓ Tomar un tiempo razonable para resolver el problema. Si no tiene éxito solicitar sugerencias o hacer el problema a un lado por un momento
- ✓ No tener miedo de volver a empezar. Suele suceder que un comienzo fresco o una nueva estrategia conducen al éxito.

1.1.2.d El mirar hacia atrás. Consiste en revisar usando el sentido común, el resultado de usar los pasos anteriores. Se deben considerar aspectos como:

- ✓ ¿Está bien el resultado?
- ✓ ¿Es la solución correcta? ¿La respuesta satisface lo establecido en el problema?
- ✓ ¿Se advierte una solución más sencilla?
- ✓ ¿Puede ver cómo extender la solución a un caso general?

EJEMPLO N°1:

A un empleado de una empresa se le cancela como Sueldo Base la cantidad de \$520.000. ¿Cuál es el sueldo líquido del empleado si además, se le cancelará un bono equivalente al 20% de su Sueldo Base?

a) Entender el problema.

Leer cuidadosamente el problema.

Se desea obtener el sueldo líquido a pago de un empleado para lo cual se necesita conocer su sueldo base, el porcentaje del bono y alguna fórmula relacionada con el problema

Identificación de datos importantes.

Sueldo Base

Porcentaje del Bono

Sueldo Líquido

b) Idear un plan.

Determinar claramente lo que hay que hacer.

Es un problema conocido

Se utilizará la fórmula

$$\text{Sueldo Líquido} = \text{Sueldo Base} + \text{Sueldo Base} * 20\%$$

Otra formula podría ser

$$\text{Sueldo Líquido} = \text{Sueldo Base} * 1,2$$

Una vez calculado, escribir el resultado.

c) Realizar el Plan

El 20% del Valor del Bono = $20/100$

Monto del Bono = $\text{Sueldo Base} * 20/100$

Sueldo Líquido = $\text{Sueldo Base} + \text{Monto del Bono}$

d) Mirar hacia atrás

Verifique: Revisar la respuesta.

Sueldo base = 520.000

Monto del Bono = $520.000 * 20/100 = 104.000$

Sueldo Líquido = $520.000 + 104.000 = 624.000$

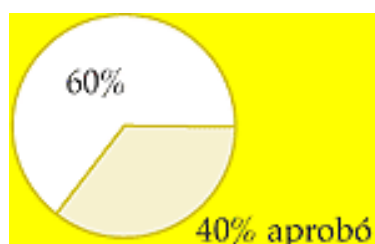
Respuesta: El sueldo líquido del empleado es de \$624.000

EJEMPLO N°2:

En una clase de 35 alumnos aprobaron el 40%. Determinar el número de alumnos reprobados.

a) Entender el problema.

Trazar un diagrama.



Identificación de datos importantes.

Total de alumnos: 35. Representan el 100%.

Total alumnos aprobados = 40% del Total de alumnos

b) Idear un plan.

Determinar claramente lo que hay que hacer.

Es un problema conocido

Al examinar el problema, lo podemos descomponer. El enunciado expresa que hay que determinar el número de alumnos reprobados, pero como sabemos que los aprobados y los reprobados representan la totalidad del curso, podemos resolver el problema de dos formas.

✓ Forma 1. Transformar el 40% de aprobados en número de alumnos utilizando una operación sencilla y restárselo al total de alumnos.

✓ Forma 2. Transformar el 60% de reprobados en número de alumnos.

Llevado a notación algebraica:

Para la Forma 1:

35-----100%

X-----40%

$$\text{Total de alumnos} - x = \text{Total de reprobados}$$

Para la Forma 2:

35-----100%

X-----60%

$$x = \text{Total de reprobados}$$

c) Realizar el Plan

Para la Forma 1: $X = (35 \times 40) / 100 = 1.400 / 100 = 14$

14 alumnos representan el 40% de los alumnos aprobados.

Por lo tanto, $35 - 14 = 21$ **alumnos reprobados**

Para la Forma 2: $X = (35 \times 60) / 100 = 2.100 / 100 = 21$

d) Mirar hacia atrás

Sumando los alumnos aprobados y reprobados debemos obtener el total de alumnos del curso. En efecto:

$$21 \text{ alumnos reprobados} + 14 \text{ alumnos aprobados} = 35 \text{ alumnos en el curso.}$$

1.1.3 Algunas sugerencias hechas por quienes tienen éxito en resolver problemas:

Además del Método de Cuatro Pasos de Polya nos parece oportuno presentar en este apartado una lista de sugerencias hechas por personas exitosas en la solución de problemas:

1. Acepta el reto de resolver el problema.
2. Rescribe el problema en tus propias palabras.
3. Tómate tiempo para explorar, reflexionar, pensar...
4. Habla contigo mismo. Hazte cuantas preguntas creas necesarias.
5. Si es apropiado, trata el problema con números simples.
6. Muchos problemas requieren de un período de incubación. Si te sientes frustrado, no dudes en tomarte un descanso -el subconsciente se hará cargo-. Después inténtalo de nuevo.
7. Analiza el problema desde varios ángulos.
8. Revisa tu lista de estrategias para ver si una o más te pueden ayudar a empezar
9. Muchos problemas se pueden resolver de distintas formas: sólo se necesita encontrar una para tener éxito.
10. No tengas miedo de hacer cambios en las estrategias.
11. La experiencia en la solución de problemas es valiosísima. Trabaja con todos tus conocimientos, tu confianza crecerá.
12. Si no estás progresando mucho, no vaciles en volver al principio y asegurarte de que realmente entendiste el problema. Este proceso de revisión es a veces necesario hacerlo dos o tres veces, ya que la comprensión del problema aumenta a medida que se avanza en el trabajo de solución.
13. Siempre, siempre mira hacia atrás: Trata de establecer con precisión cuál fue el paso clave en tu solución.
14. Ten cuidado en dejar tu solución escrita con suficiente claridad de tal modo que puedas entenderla si la lees 10 años después.
15. Ayudar a que otros desarrollen habilidades en la solución de problemas es una gran ayuda para uno mismo: No les des soluciones; en su lugar provéelos con sugerencias importantes.
16. ¡Disfrútalo! Resolver un problema es una experiencia fantástica.

OBJETIVO 1.2: ANALIZAR UN PROCESO COMPUTACIONAL EN TÉRMINOS DE ENTRADA – OPERACIÓN INTERNA – SALIDA

1.2.1 CONCEPTO DE DATO E INFORMACIÓN

Datos: Es todo aquella representación de una entidad y que es susceptible de tratamiento ya sea en un programa o proceso informático. Por ejemplo nombre, apellido y edad son datos de una persona (entidad).

En otras palabras un dato es la representación de una realidad.

Información: Mensaje válido para un receptor o resultado del procesamiento de datos.

Computador: Máquina capaz de aceptar datos de entrada, procesarlos y entregar resultados de salida (información).

Esquema general de un computador



Programa (software): Conjunto de órdenes o instrucciones capaces de manipular un conjunto de datos. Estas órdenes pueden ser divididas en tres grandes bloques claramente diferenciales, estos son:

- **Entrada de datos:** En este bloque se engloban todas aquellas instrucciones que toman datos de un dispositivo o periférico externo, depositándolos en la memoria principal del computador para ser procesados.
- **Proceso:** Engloban todas aquellas instrucciones encargadas de modificar los datos que previamente habían sido depositados en la memoria principal. Todos los resultados obtenidos en el tratamiento de dichos datos son depositados nuevamente en la memoria principal quedando de esta manera disponible.
- **Salida de resultados:** Es el conjunto de instrucciones que toman los resultados finales desde la memoria principal y lo envían a dispositivos externos.

Ejercicios propuestos

1. Crear grupos de trabajos de 5 integrantes. Los alumnos deberán determinar si existen datos de entrada y cuáles son, qué procesos se realizan sobre ellos y cuáles son los datos de salida.

- Determinar el promedio de edad entre los integrantes del grupo.
- Determinar la cantidad de mujeres y la cantidad de hombres dentro del grupo.
- Contar los alumnos que son mayores de edad (18 años o más) y menor de edad (inferior a 18 años), en el grupo.
- Determinar la cantidad de integrantes que salieron de cuarto medio con un promedio igual o inferior a 5.0.

Solución esperada

- 1a. Datos de entrada = Cantidad de personas. Edades de las personas
Procesamiento = Sumar las edades y dividirla por la cantidad de personas.
Datos de salida = Promedio de edad de todas las personas del grupo.
- 1b. Datos de entrada = Personas del grupo.
Procesamiento = Contar las mujeres. Contar los hombres.
Datos de salida = Cantidad de mujeres y hombres.
- 1c. Datos de entrada = Edades de los alumnos del grupo.
Procesamiento = Cada edad comparar con 18. Contar los menores a 18 años. Contar los que tienen 18 o más años en forma separada.
Datos de salida = Cantidad de mayores de edad. Cantidad de menores de edad.
- 1d. Datos de entrada = Notas de egreso de 4° Medio
Procesamiento = Cada nota comparar con 5.0. Contar las menores o iguales a 5.0.
Datos de salida = Cantidad de personas que salieron de 4° Medio con nota promedio igual o menor a 5.0
2. Para las siguientes situaciones identificar: datos, proceso y resultado (información).
 - a. El promedio de las edades de los 30 alumnos de un curso es de 22 años.

 Datos: _____
 Proceso: _____
 Información: _____
 - b. En una agroindustria a un productor por 2 toneladas de harina se le cancelaron 4 millones de pesos.

 Datos: _____
 Proceso: _____
 Información: _____
 - c. Para cercar un jardín de 3 metros de largo por 5 metros de ancho se necesitaron 16 metros de alambre.

 Datos: _____
 Proceso: _____
 Información: _____

UNIDAD 2: ALGORITMOS Y ESTRUCTURAS DE CONTROL

OBJETIVO 2.1: DESCRIBIR EL PROCESO DE CREACIÓN DE UN PROGRAMA PARA PROBLEMAS SIMPLES

2.1.1 PROCESOS DE CREACIÓN DE PROGRAMAS.

Programa: Conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema específico. Todo programa debe pasar por fases de elaboración hasta llegar al producto final.

Resolvemos problemas a diario, pero normalmente ignoramos el proceso que estamos siguiendo. La mayoría de nuestra experiencia es simplemente seguir algoritmos, recetas. Por ejemplo: practicamos un juego, preparamos café, caminamos, etc.

Etapas en el proceso de resolución de problemas

Para aprender a programar hay que desarrollar en forma consciente alguna estrategia de resolución de problemas.

Algunas de las etapas que se deben seguir en este proceso son:

1. **Diseño del proceso:** Esta etapa consiste en el análisis del problema, esto es: entender el problema, determinar estrategias de solución. En ambos casos se indicará claramente los procesos a seguir (lo que se va a hacer) para llegar a la solución correcta. En esta etapa además, se desarrolla un modelo que no es más que una representación apropiada al problema planteado. En otras palabras, se debe visualizar como se realizará la transformación de los datos de entrada para que a través de un proceso apropiado se obtenga la salida correcta.
2. **Construcción:** Esta etapa consiste en desarrollar un algoritmo que pueda representar la estructura del programa la cual puede ser en forma narrativa, diagramas de flujo o Pseudolenguaje.
3. **Codificación:** Consiste en la traducción del algoritmo a algún lenguaje de programación, el cual crea instrucciones entendibles y ejecutables por el computador; algunos de estos lenguajes pueden ser: C, C++, Visual Basic, Java, Cobol, Pascal, etc.
4. **Verificación:** Se prueba el algoritmo en forma analítica para demostrar su efectividad; vale decir, se prueba el algoritmo y/o programa con datos conocidos que producirán una salida conocida. Con esto se pretende detectar y corregir los errores para mejorar el algoritmo.
5. **Documentación:** En esta etapa se debe documentar el o los programas realizados en la etapa de construcción. Se debe confeccionar un manual de uso que contenga, al menos los siguientes puntos:
 - ✓ Descripción del método de solución utilizado
 - ✓ Requerimientos lógicos (Sistema Operativo, Software, etc.).
 - ✓ Requerimientos físicos (Servidores, computadores, periféricos, etc.)

Características de un programa

Para asegurar la calidad de los programas y sistemas, es de vital importancia, considerando que para un problema existen muchas soluciones, elegir la alternativa más eficaz y eficiente en función del problema dado. Todo programa debe cumplir al menos con las siguientes características:

- a) **Legibilidad:** Claro y sencillo para una fácil lectura y comprensión.
- b) **Modificabilidad:** El programa para su vigencia y actualización debe ser de fácil mantenimiento.
- c) **Fiabilidad:** Debe ser robusto, fácil de recuperarse frente a errores o malos usos
- d) **Eficiencia:** Eficiente, aprovechando al máximo los recursos.
- e) **Portabilidad:** El programa debe ser de fácil codificación para distintos lenguajes.

Ejercicios propuestos

Para las siguientes situaciones identifique la etapa o fase del proceso en la resolución de un programa.

- a. El algoritmo creado se traducirá en lenguaje C++.
Etapa _____
- b. Algoritmo requiere de una formula que calcule el volumen de un prisma.
Etapa _____
- c. La resolución de un problema ha sido escrita en forma narrativa.
Etapa _____
- d. Un algoritmo ha sido recorrido con datos conocidos.
Etapa _____

Solución esperada

- a. Etapa de Codificación
- b. Diseño
- c. Construcción
- d. Verificación

OBJETIVO 2.2: EXPLICAR LOS PRINCIPIOS QUE RIGEN EL ALMACENAMIENTO DE DATOS

2.2.1 PRINCIPIOS DE ALMACENAMIENTO

La memoria del computador se divide en dos:

- ✓ Memoria Central o Interna
- ✓ Memoria Auxiliar o Externa

Memoria Central o interna: La CPU utiliza la memoria del computador para guardar información mientras trabaja. Mientras esta información permanece en memoria, el computador puede tener acceso a ella en forma directa. La memoria interna consta de dos áreas de memoria:

- **La memoria RAM (Random Access Memory):** Recibe también el nombre de memoria principal o memoria de usuario, en ella se almacena información sólo mientras el computador este encendido. Cuando el computador se apaga o arranca nuevamente la información se pierde, por lo que se dice que la memoria RAM es una memoria volátil.
- **La memoria ROM (Read Only Memory):** Es una memoria estática que no puede cambiar, el computador puede leer los datos almacenados en la memoria ROM, pero no se pueden introducir datos en ella, o cambiar los datos que ahí se encuentran; por lo que se dice que esta memoria es de sólo lectura (Read Only). Los datos de la memoria ROM están grabados en forma permanente y son introducidos por el fabricante del computador.

Memoria Auxiliar (Externa): Es donde se almacenan todos los programas o datos que el usuario desee. Los dispositivos de almacenamiento o memorias auxiliares (externas o secundarias) más comúnmente utilizados son: cintas magnéticas, discos magnéticos discos ópticos, disquetes.

Dirección y contenido de Memoria:

La dirección de memoria es un número que identifica un lugar dentro de la memoria. El contenido es lo que se encuentra dentro de una dirección.

Como analogía podemos pensar que dirección de memoria se asemeja al número que identifica a las casillas de correos y el contenido es lo que tienen adentro cada una de esas casillas.

Ejercicios:

- La siguiente ilustración es una representación conceptual de una memoria, cuyas direcciones están representadas por variables (X, R, Y, etc) y los contenidos son los que se muestran (3,12,4,-9,etc)

X	R	Y	S	L	P
3	12	4	-9	100	8
T	G	A	J	W	Z
100	6	20	5	80	5

- Determinar los valores finales de los contenidos si se ejecutan secuencialmente las siguientes operaciones aritméticas.

$X := G + 5$
 $W := A * X$
 $T := S + P - J$
 $J := T + L$
 $Z := P * R + Y$
 $Y := Y + Z$
 $Y := Y + 1$

- Escriba los valores finales de las variables

X	R	Y	S	L	P
T	G	A	J	W	Z

- La siguiente ilustración es una representación conceptual de una memoria, cuyas direcciones están representadas por variables (X, R, Y, etc.) y los contenidos son los que se muestran.

X	R	Y	S	L	P
14	8	0	-9	130	0
T	G	A	J	W	Z
16	-56	-4	5	-8	0

- a) Determinar los valores finales de los contenidos si se ejecutan secuencialmente las siguientes operaciones matemáticas.

$X := X + A - W$
 $S := X - J * Z$
 $R := R + 1$
 $Y := R - X + 2$
 $L := Y + L$
 $T := G + J - Z * 3$
 $W := W + 5$
 $J := J + Z / X + 2$

- b) Escriba en la siguiente ilustración los valores finales de las variables

X	R	Y	S	L	P
T	G	A	J	W	Z

3. Para las siguientes situaciones identifique la etapa o fase del proceso en la resolución de un programa.

- a) Para un problema dado, ¿que haremos para llegar a una solución correcta?

Etapa _____

- b) Al recorrer el algoritmo, algunos datos el resultado no son los esperados:

Etapa _____

- c) Un algoritmo será traducido a un lenguaje de programación:

Etapa _____

- d) Un algoritmo calcula el área de un cuadrado para un valor cualquiera de un lado:

Etapa _____

- e) Al parecer el modelo matemático escogido soluciona el problema sólo para un tipo de datos:

Etapa _____

OBJETIVOS 2.3: DESCRIBIR A TRAVÉS DE DEFINICIONES Y EJEMPLOS CADA UNO DE LOS ELEMENTOS QUE COMPONEN UN ALGORITMO

2.3.1 CONCEPTO DE ALGORITMO

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe *alkhowarizmi*, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir para dar solución a un problema específico.

Tipos de Algoritmos

Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.

Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

Lenguajes Algorítmicos. Es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso. Hay dos tipos:

Gráficos: Es la representación gráfica de las operaciones que realiza un algoritmo. Ejemplo: Diagrama De Flujo, DFD.

No Gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo. Ejemplo: Pseudocódigo.

2.3.2 ESTRUCTURA DE UN ALGORITMO.

Las características de un buen algoritmo son:

- ✓ Debe tener un punto único de inicio.
- ✓ Debe ser definido, no debe permitir dobles interpretaciones.
- ✓ Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
- ✓ Debe ser finito en tamaño y tiempo de ejecución. Es decir, debe tener un fin.

Ejercicios:

Presentar y discutir el siguiente ejercicio:

Se desea construir un algoritmo que cambie la rueda de un automóvil. La siguiente lista de paso le ayudará en la construcción del algoritmo pero, para ello debe ordenarlas en una secuencia lógica.

- | | |
|---------------------------------------|---------------------------------|
| a) Sacar las tuercas | b) Inicio |
| c) Bajar el auto | d) Colocar rueda de repuesto |
| e) Sacar herramientas y repuesto. | F) Cerrar maletera |
| g) Colocar tuercas | h) Fin |
| i) Abrir maletera | j) Apretar tuercas |
| k) Posicionar gata y levantar el auto | l) Guardar herramientas y rueda |
| m) Retirar rueda mala | n) Soltar tuercas |

Solución: b, i, e, f, n, k, , a, m, d, g, j, c, i, l, f, h

Presentar y discutir el siguiente ejercicio:

Solicitar que un alumno del curso haga la representación de un robot. Otro alumno le dará instrucciones para caminar desde su puesto hacia la puerta de salida de la sala.

Algunos supuestos:

- ✓ El alumno robot escucha y entiende instrucciones
- ✓ El proceso “mover el pie” implica todos los subprocesos involucrados como, levantar la pierna, flectar las rodillas, mover dedos, etc.

Las instrucciones deberán ser del tipo:

- ✓ Mover pie derecho hacia delante
- ✓ Mover pie izquierdo hacia delante
- ✓ Detenerse
- ✓ Girar izquierda x grados
- ✓ Girar derecha x grados

Dejar como tarea o resolver en clases algunos de los siguientes problemas cotidianos.

- a. Cambiar las cuatro ruedas de un vehículo.
- b. Preparar una taza de café
- c. Tomar la micro desde la casa hasta la sede.



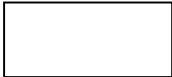
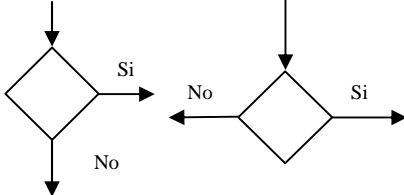
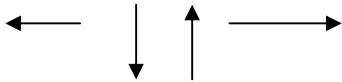

OBJETIVO 2.4: REPRESENTAR GRÁFICAMENTE LA CIRCULACIÓN DE DATOS E INFORMACIÓN Y LA SECUENCIA DE OPERACIONES INVOLUCRADAS EN UN ALGORITMO DADO.

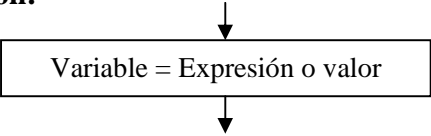
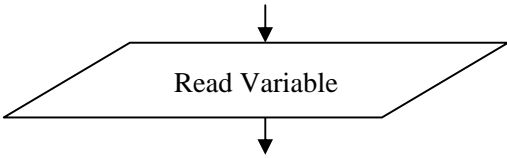
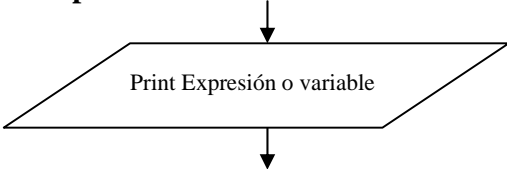
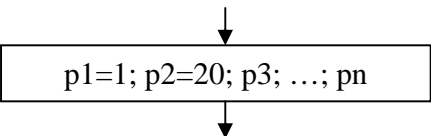
2.4.1 DEFINICIÓN DE DIAGRAMA DE FLUJO

Para el diseño de algoritmos se utilizan técnicas de representación. Una de estas técnicas son los Diagramas de Flujo (DDF), que se definen como la representación gráfica que mediante el uso de símbolos estándar unidos mediante líneas de flujo, muestran la secuencia lógica de las operaciones o acciones que debe realizar un computador, así como la corriente o flujo de datos en la resolución de problema.

2.4.2 SIMBOLOGÍA DE DIAGRAMA DE FLUJO

Algunos símbolos que se utilizan para desarrollar algoritmos en DDF son:

Símbolo	Función
	Terminal. Marca el inicio y/o el final en la ejecución de un DDF
	Operación de E/S en general. Se utiliza para la introducción de datos desde un periférico a la memoria del computador y para la salida de resultados desde la memoria del computador a un periférico.
	Proceso en general. Utilizado para mostrar cualquier tipo de operación durante el proceso de elaboración de los datos depositados en la memoria.
	Decisión de dos salidas, indica operaciones lógicas o comparativas seleccionando la ruta en función del resultado (si, no).
	Indicadores de la dirección del flujo de datos
	Conector. Este símbolo es utilizado para el reagrupamiento de líneas de flujo.

OPERACIONES BÁSICAS PARA ALGORITMOS	
Asignación: Permite depositar valores o resultados de expresiones en una variable. Variable := Expresión o valor	Asignación: 
Leer Variable: Toma uno o varios datos desde un dispositivo de entrada para almacenarlos en variables Read Variable	Leer Variable: 
Escribir Expresión: Imprime datos en los dispositivos externos, como impresora o pantalla Print Expresión o variable	Escribir Expresión : 
Procesos: Instrucciones que modifican las variables a partir de un estado inicial hasta un estado final. p1:=1; p2:=20; p3; ...; pn	Procesos : 

Algunas recomendaciones para el diseño de Diagramas de Flujo

1. No deben quedar líneas de flujo sin conectar.
2. Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
3. Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.
4. Se deben usar solamente líneas de flujo horizontales y/o verticales.
5. Se debe evitar el cruce de líneas utilizando los conectores.
6. Se deben usar conectores sólo cuando sea necesario.

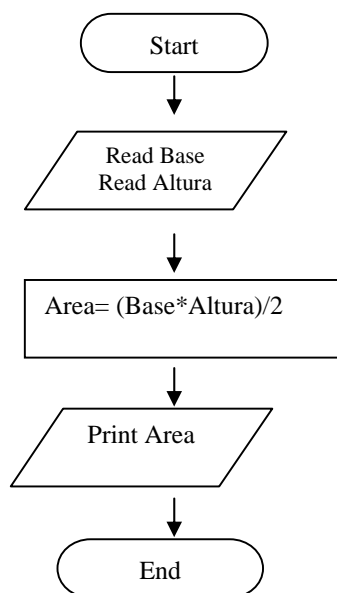
Ejemplos de desarrollo de algoritmos utilizando Diagramas de Flujo

Ejemplo N°1

Algoritmo que permite calcular el área de un triángulo. Recordemos que la formula es:

$$\text{Área} = (\text{Base} * \text{Altura}) / 2.$$

Solución

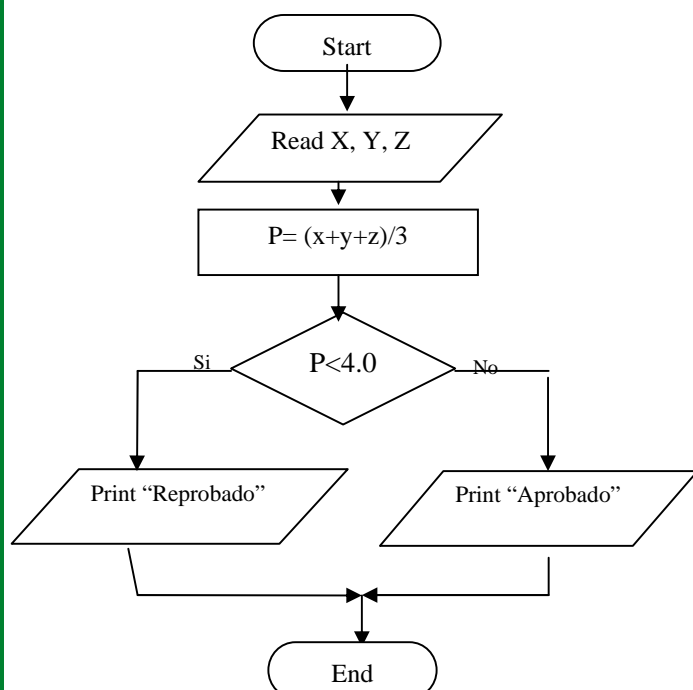


Para verificar el diagrama de flujo utilizamos datos de prueba. Consideremos Base = 5 y Altura = 8. El resultado es la impresión del valor de la variable Área que es 20

Ejemplo N°2

Algoritmo que captura tres notas y saca el promedio. Si el promedio es menor a 4.0 imprimir “Reprobado” caso contrario imprimir “Aprobado”

Solución

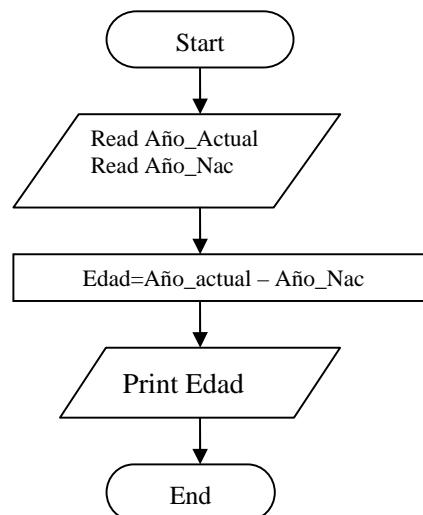


Para verificar que DDF está lógicamente bien construido utilizamos datos de prueba. Consideremos X=4,5; Y= 5,0 ; Z=5,5. La suma es 15, por lo tanto en P queda un 5,0. Se imprimirá la palabra o string “Aprobado”

Ejemplo N°3

Algoritmo que calcula la edad de una persona conociendo el año de nacimiento y el año actual.

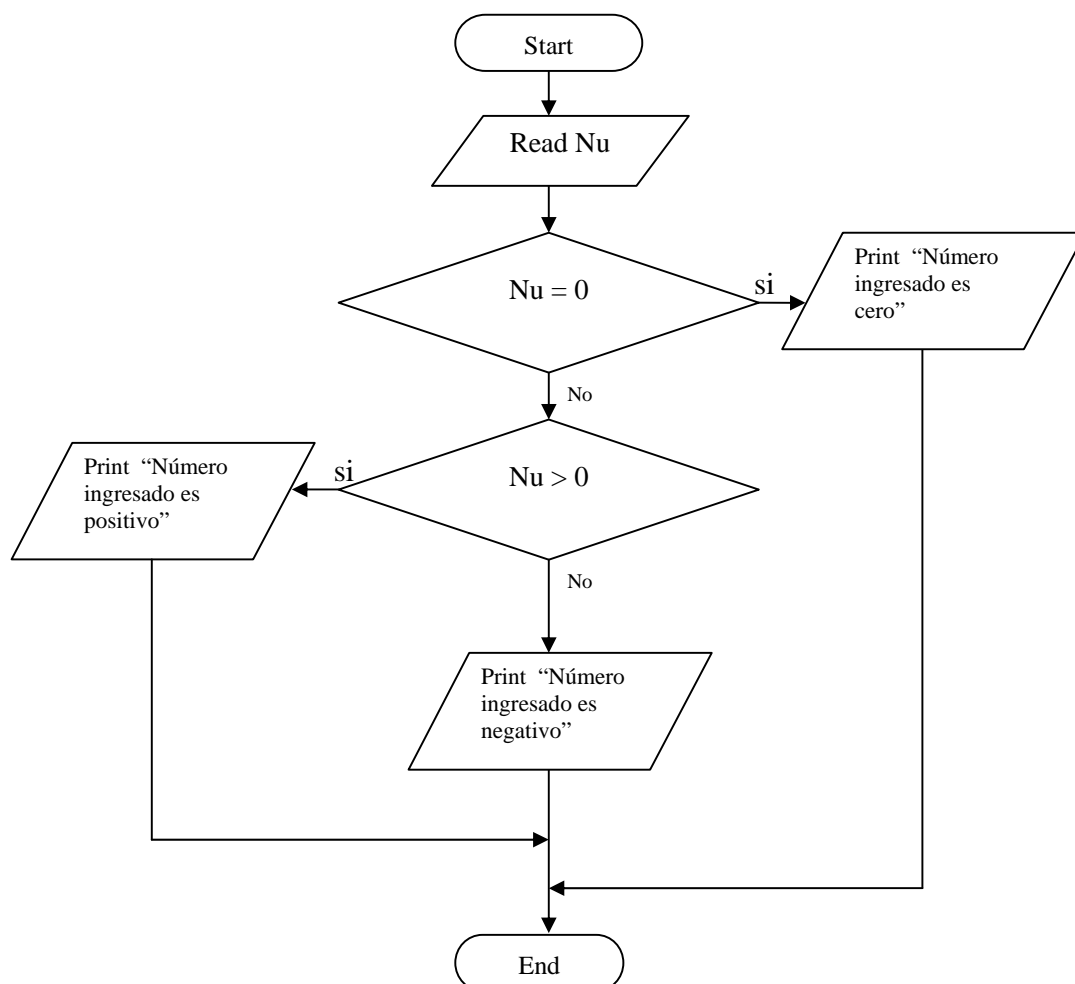
Solución



Ejemplo Nº4

Hacer un DDF que verifique si un número es positivo, negativo o cero.

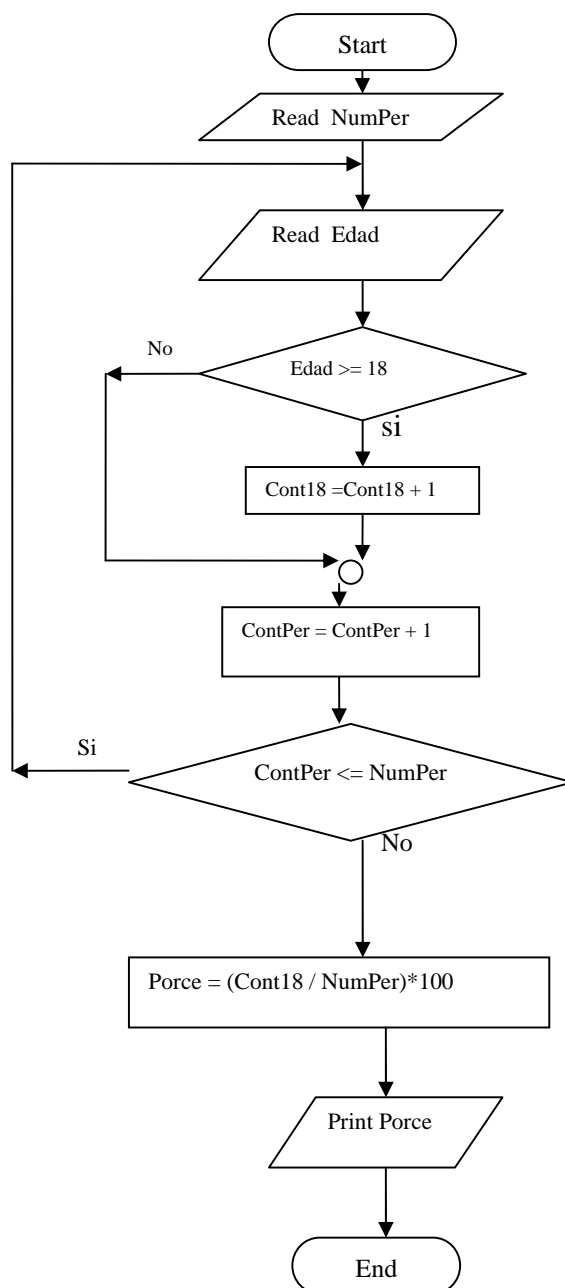
Solución



Ejemplo N°5

Hacer un DDF para determinar el porcentaje de personas, de un grupo, que son mayores de edad (edad mayor o igual a 18 años).

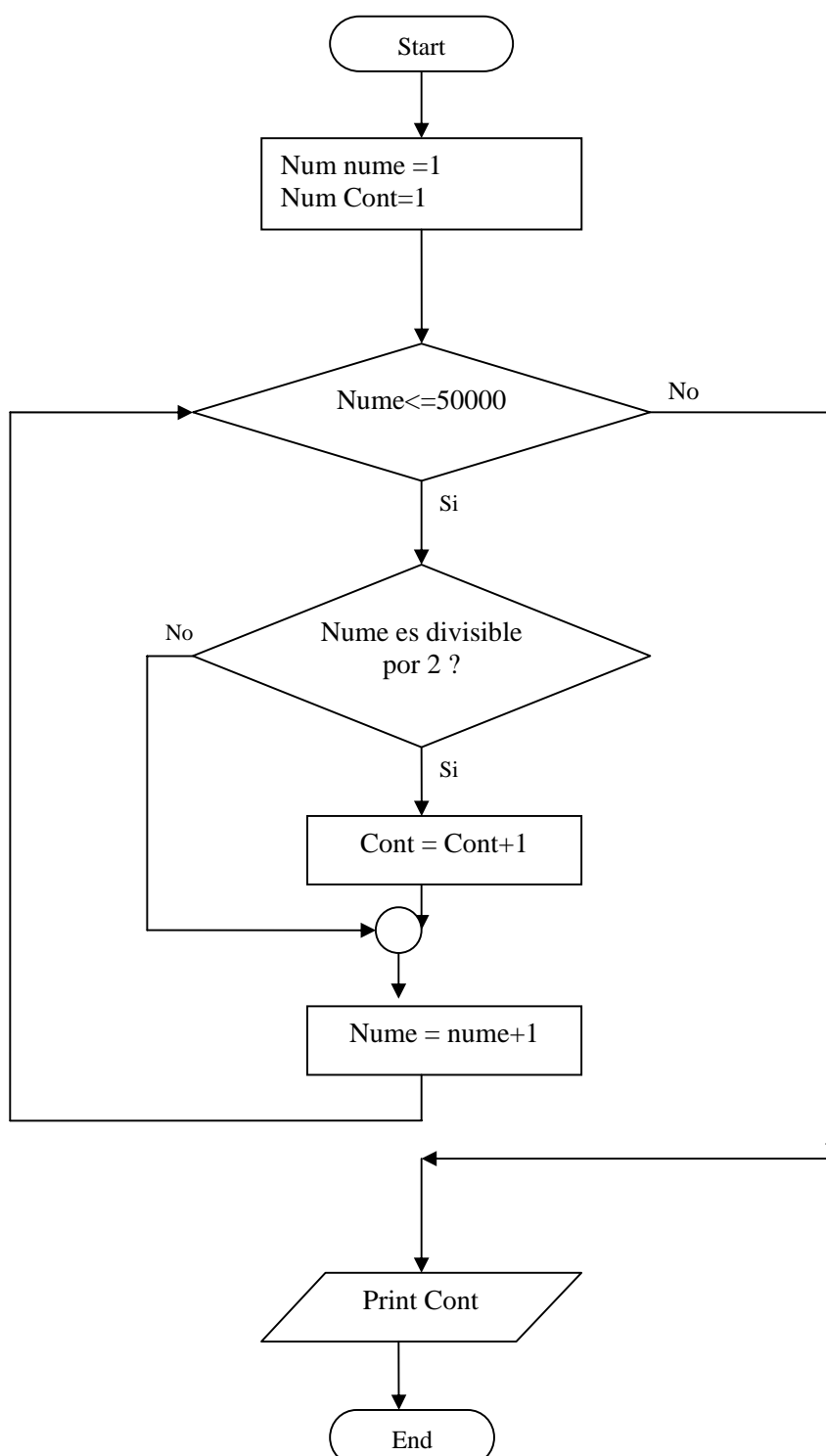
Solución



Ejemplo N°6

Hacer un DDF para determinar la cantidad de números pares en los primeros 50.000 números naturales. (Incluidos el 1 y el 50.000). Nótese que aquí se incorpora el concepto de declaración de variable e inicialización

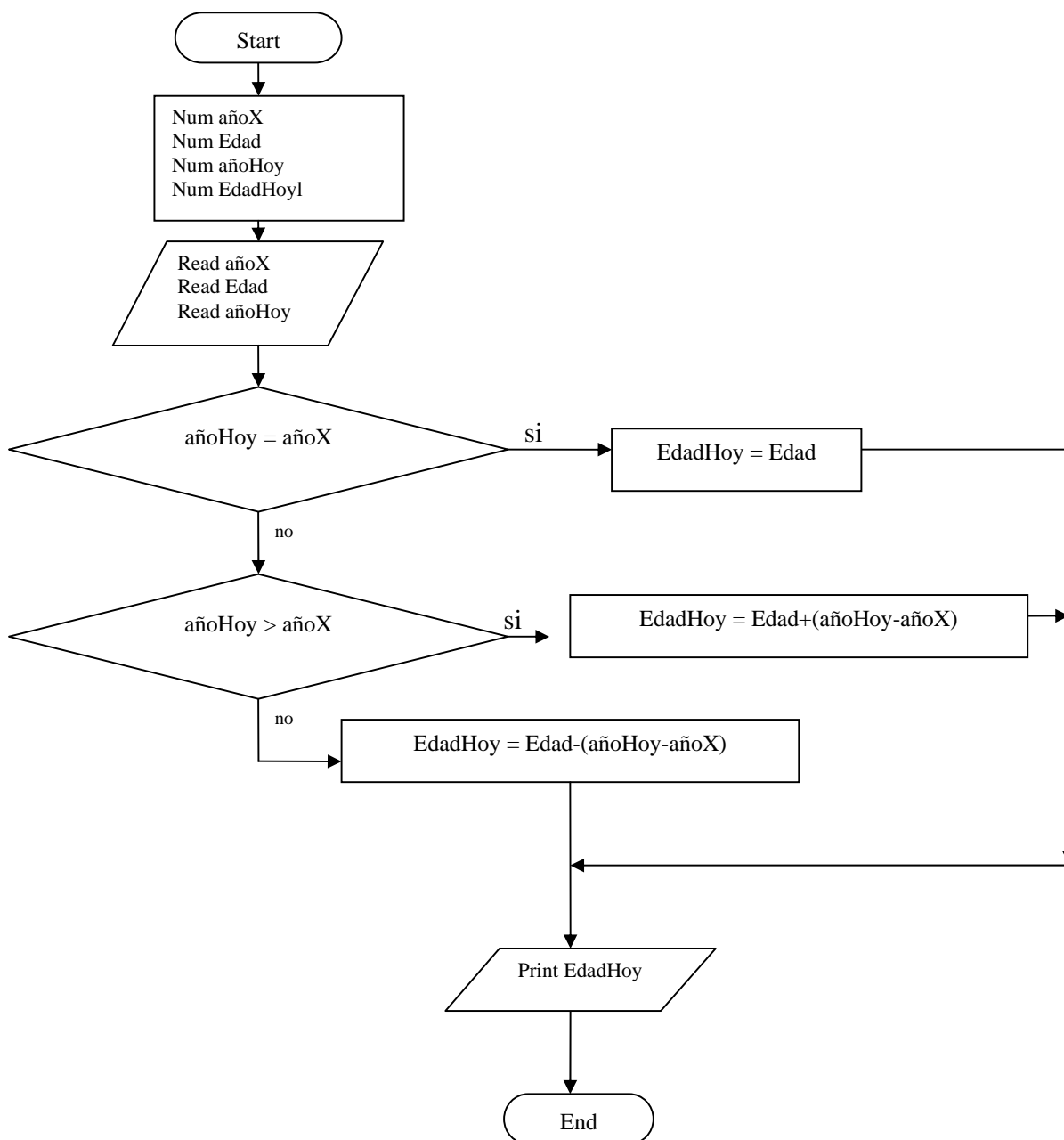
Solución



Ejemplo N°7

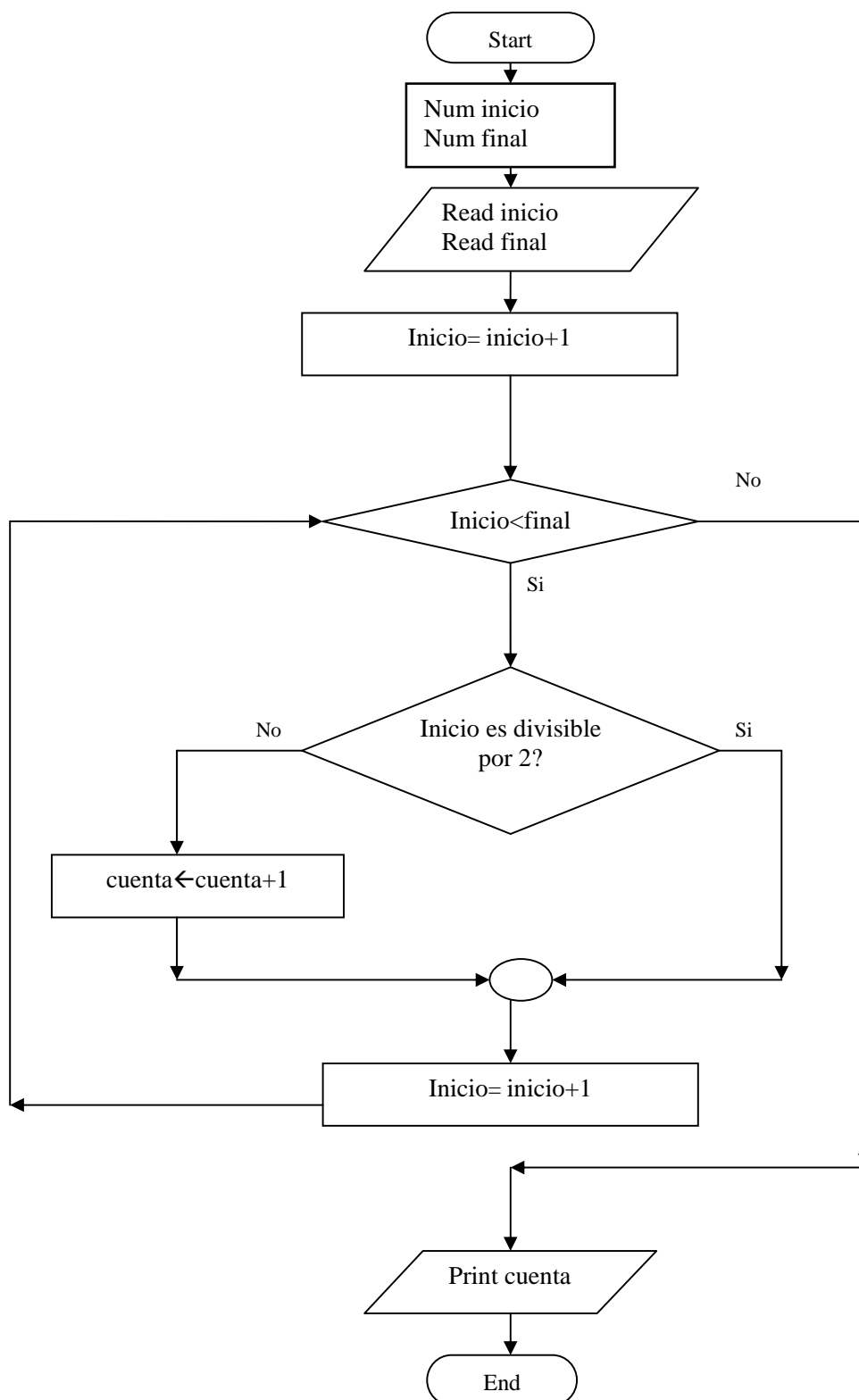
Hacer un DDF que permita calcular la edad actual de una persona teniendo como referencia la edad de la persona en otro año. Por ejemplo, saber que edad actual tiene una persona sabiendo que para el 2010 tendrá 9 años. Se debe ingresar también el año actual

Solución



Ejemplo N°8

Hacer un DDF que permita determinar la cantidad de números impares entre dos números naturales dados. No debe incluir los números dados.

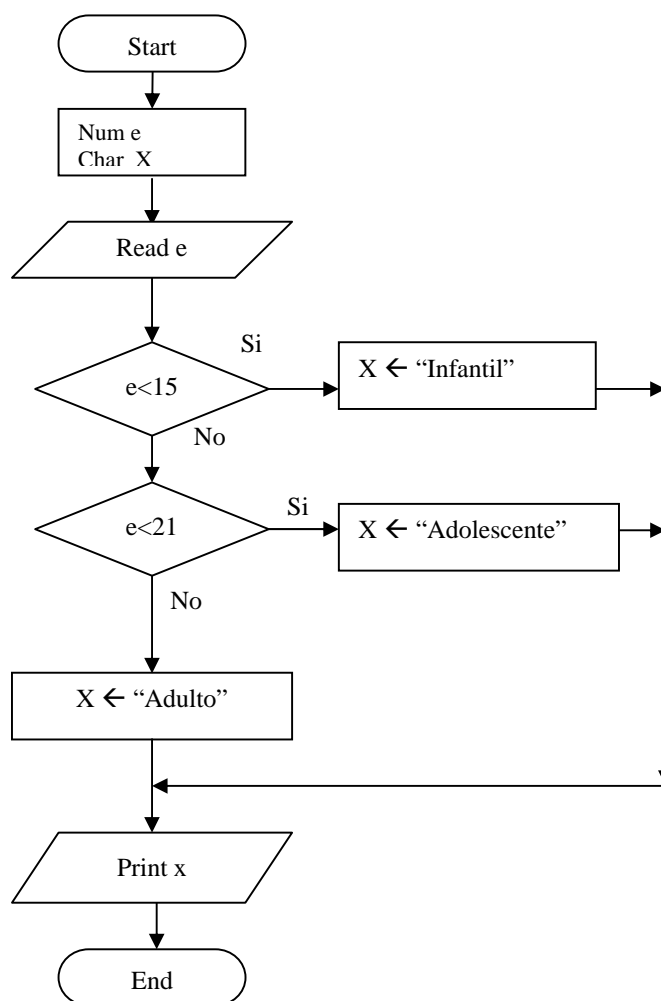


Ejercicios propuestos (OBJETIVO 2.4)

- 1.- Crear un algoritmo DDF que al leer un número entero positivo (asuma que el número cumple las condiciones), imprima la palabra “PAR” si el número es par e “IMPAR” si es impar.
- 2.- Supongamos que la variación del IPC de los meses de Febrero y Marzo fueron 0.3% y 0.6% respectivamente. Crear un algoritmo en DDF que muestre el valor de un producto actualizado al 31 de marzo y la diferencia de precio entre el mes de Febrero y Marzo. Precio de referencia al 1 de febrero.
- 3.- Confeccione un algoritmo en diagrama de flujo que al leer el neto de una factura, calcule el IVA y de cómo salida el total de la factura.
- 4.- Crear un algoritmo en diagrama de flujo que al ingresar dos números imprima el mayor de ellos o IGUALES si son iguales.
- 5.- Confeccionar un algoritmo en diagrama de flujo que imprima el salario reajustado de un trabajador según las siguientes tabla:

Categoría	Porcentaje Reajuste
Cat1	20%
Cat2	15%
Cat3	10%
Cat4	8%

6. Recorrer el siguiente DDF e indicar que es lo que hace



Haga el ruteo con los siguientes valores.

e	Salida
28	
15	
21	
14	

OBJETIVO 2.5: DESCRIBIR CADA UNO DE LOS ELEMENTOS QUE COMPONEN UN ALGORITMO EN PSEUDOLENGUAJE

2.5.1: DEFINICIÓN DE PSEUDOLENGUAJE

Los DDF presentan algunos inconvenientes:

- ✓ Cualquier modificación obliga a reorganizar el diagrama de nuevo.
- ✓ Utiliza técnica lineal, es desuso hoy día.
- ✓ El proceso de recorrer un algoritmo desde el inicio hasta el final puede ser complejo.

Para dar solución a los problemas que presentan los DDF se utiliza el pseudocódigo o pseudolenguaje: Un Pseudocódigo describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de lenguaje de programación y palabras claves.

Antes de usar la técnica de pseudolenguaje para la resolución de problemas, es necesario conocer una serie de conceptos.

2.5.2: TIPOS DE DATOS

Recordemos que dato es un tipo especial de información no elaborada. En general valores sin características asociadas.

Puesto que los datos los utiliza el computador para su procesamiento, según sea lo que se desee guardar como dato, tenemos:

- Numéricos: con ellos se puede realizar operaciones aritméticas. Pueden ser:
 - a) enteros: son todos los positivos y negativos
Ejemplo: -23 4 0 -23 33
 - b) reales incluye a los números decimales
Ejemplo: -0,234 -1,1 3,14 345,8
- Alfanuméricos: Corresponden a las letras mayúsculas y minúsculas, caracteres especiales como guiones, paréntesis, signos de puntuación, etc.; cuando se definen números de esta forma no se pueden efectuar operaciones aritméticas. También se pueden utilizar mezclas de letras, caracteres y números.
Ejemplo: alpha-56 rwy-45_3-s

Para los efectos de desarrollar algoritmos en pseudolenguaje, se utilizará solamente variables del tipo numérica o alfanumérica

2.5.3: VARIABLES

Se considera variable a una zona de memoria referenciada por un nombre donde se puede almacenar el valor de un dato, que puede cambiarse cuando se desee. El nombre de la variable es elegido por el usuario pero debe seguir ciertas reglas.

Se debe tener muy claro que una variable no es un dato, sino un área de memoria que contendrá un dato.

A cada variable, el computador le asigna una dirección de memoria. Cuando se haga referencia a esa variable el computador irá siempre a esa dirección.

Para que una variable quede correctamente definida se debe especificar:

- **Tipo**
- **Nombre**
- **Valor**

Tipo. El tipo de variable depende del tipo de datos que va a contener. Por lo tanto habrá tantos tipos de variables como tipos de datos. Por ejemplo: reales, enteras, simple precisión, doble precisión, alfanumérico, booleano, etc. En este curso utilizaremos solamente variables del tipo numérica y alfanumérica.

- **Variables del tipo numéricas:**

Sintaxis: num nombre_variable := número

El prefijo num indica que es una variable del tipo numérica

El símbolo := indica asignación

Ejemplo:

```
num a:=1
num código := 567
num contador:= -12
```

- **Variables del tipo Alfanuméricas:**

Sintaxis: char nombre_variable[numero de filas] (numero de caracteres)

El prefijo char indica que es una variable del tipo alfanumérica

char nombre (5), la variable nombre puede contener hasta cinco caracteres

Si el largo de la variable es mayor que lo asignado, rellena con blancos

Char nombre (15)

nombre := "Felipe", la variable nombre contiene : Felipe más 9 blancos

Ejemplos:

```
Char b (4)
b:= "hola"
char apellido (7)
apellido := "hoffman"
char resp (1)
resp:= "y"
char blancos (1)
blancos := " "
```

Las cremillas no se almacenan, sólo su contenido.

Ejemplo

Char nombre [10] (5), el vector llamado *nombre* podrá contener 10 elementos de largo 5 caracteres.

Nombre. El nombre de las variables es muy dependiente del lenguaje utilizado puesto que entre otros, existen palabras reservadas que no pueden ser utilizadas como nombres de variables.

Valor. Indica cual es el valor que va tomar la variable en un principio. También se conoce como inicialización. Puede omitirse.

2.5.4: CONSTANTES

Es una posición de memoria, referenciada por un nombre, donde se almacena un valor que no puede cambiar y permanece invariable a lo largo del proceso. Tiene las mismas características de las variables en cuanto a nombre, tipo y valor.

Ejemplos de constantes:

```
num circulo := 360
num an_recto := 90
num pi := 3.1416
char país := "chile"
char fono := "2296566"
```

2.5.5: OPERADORES

Todos los símbolos que enlazan argumentos en una operación se llaman operadores y se utilizan para construir expresiones.

Ejemplo: $a + b = c$ aquí: el signo $+$ y el signo $=$ son operadores.

Los operadores pueden ser:

Aritméticos. Para tratar con números se utilizan los operadores aritméticos, de acuerdo a:

SIGNO	SIGNIFICADO
+	Suma
-	Resta
*	Multiplicación
^	Potenciación
/	División real
DIV	División entera
MOD	Resto de la división

Operadores aritméticos

Los operadores aritméticos permiten la realización de operaciones matemáticas con valores, variables y constantes.

Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.

Ejercicios:

- | Operación | Resultado |
|-----------|-----------|
| 12*12 | 144 |
| 12^2 | 144 |
| 123 DIV 4 | 30 |
| 22 MOD 5 | 2 |
- Indicar el valor de x en cada uno de los siguientes casos:
 - $x := (2+3)*6$
 - $x := (12+6)/2*3$
 - $x := (2+3)/4$
 - $x := (2+3)MOD4$
 - $x := (2+3)DIV4$
 - $x := (3*4+2)*(15DIV2)$
 - $x := 2^2+3-2*(5MOD2)$
 - $x := 3*6*2/8-3*(19DIV6)$
- Expresar en formato computacional utilizando los operadores aritméticos las siguientes expresiones:

a) $\frac{m+n}{n}$ b) $\frac{m+n/p}{p-r/s}$ c) $\frac{m+4}{p-q}$ d) $\frac{crt}{100}$

Alfanumérico

Se utilizan para unir datos alfanuméricos. Se llama también: concatenar.

SÍMBOLO	SIGNIFICADO
+	Concatenación

Operador alfanumérico

Concatenación: unir expresiones alfanuméricas como eslabones de una cadena.

Ejemplo N°1

Expresión	Resultado
"pseudo" + "código"	"pseudocódigo"
"3" + "." + "1416"	"3.1416"

Ejemplo N°2

char a, b, c		
a:="hola, "	b:="como te "	c:="va"
a+b+c --> "hola, como te va"		

Relacionales

Se usan para formar expresiones booleanas, es decir expresiones que al ser evaluadas producen un valor booleano: verdadero o falso. Se utilizan para establecer una relación entre dos valores.

SÍMBOLO	SIGNIFICADO
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
<>	Distinto que
=	Igual que

Operadores relacionales

Ejemplos:

Comparación	Resultado
12 => 21	falso
8 <> 8	falso
8 > -8	verdadero
56 >= 56	verdadero

Cuando la comparación se hace con datos alfanuméricos, opera de acuerdo a las siguientes reglas:

- ✓ Se compara uno a uno de izquierda a derecha.
- ✓ Si son de diferente longitud pero exactamente iguales hasta el último carácter del más corto, entonces el más corto es el menor.
- ✓ Sólo son iguales dos datos alfanuméricos si son iguales su contenido y su longitud.
- ✓ Las letras minúsculas tienen mayor valor que las mayúsculas. (tabla ascii)

Ejemplo:

Comparación	Resultado
"a" < "b"	verdadero
"aaaa" > "aaa"	verdadero
"b" > "aaa"	verdadero
"ab" > "aaaa"	verdadero
"c" < "c"	falso
"2" < "12"	falso

Lógicos o Booleanos

Combinan sus operandos de acuerdo al álgebra de Boole para producir un nuevo valor que se convierte en el valor de la expresión.

SÍMBOLO	SIGNIFICADO
OR	Suma Lógica
AND	Producto Lógico
NOT	Negación

Operadores lógicos

OR u O: es un operador binario, afecta a dos operadores. La expresión que forma es verdadera cuando al menos uno de sus operandos es verdadero. Es un operador de disyunción.

Ejemplo: estudiamos o vamos al estadio

AND o Y: también es un operador binario. La expresión formada es cierta cuando ambos operadores son ciertos al mismo tiempo. Es el operador lógico de conjunción

Ejemplo: si es verano y hace calor vamos a la playa

NOT o NO: es un operador unario, afecta a un solo operando. Cambia el estado lógico de la expresión; si es verdad la transforma en falso y al revés.

Ejemplo: no es verano

El orden de prioridad de estos operadores es: **NOT**, **AND** y **OR**.

Operadores Lógicos y las proposiciones: Los operadores lógicos, permiten entre otras cosas resolver proposiciones, donde llamaremos proposiciones a toda expresión que tenga un sentido y de la cual se puede decir que es verdadera o falsa. Esta es llamada lógica proposicional.

Ejemplo:

- a) La quinta región se caracteriza por ser zona minera (Esta proposición es falsa).
- b) ¿Cómo se resuelve este ejercicio? (Esta no es una proposición)
- c) El caballo blanco de Napoleón es blanco (Esta proposición es verdadera).

Las proposiciones pueden ser nominadas con algunas letras tales como p, q, r, etc. Las cuales pueden ser unidas por conectivos lógicos, dando origen a una expresión lógica.

Los operadores lógicos más utilizado en la programación es la negación NOT, los conectivos AND, OR. Su tablas de verdad son:

P	NOT P
1 (V)	0 (F)
0 (F)	1 (V)

OPERADOR NOT

P	Q	P AND Q
1 (V)	1 (V)	1 (V)
1 (V)	0 (F)	0 (F)
0 (F)	1 (V)	0 (F)
0 (F)	0 (F)	0 (F)

OPERADOR AND

P	Q	P OR Q
1 (V)	1 (V)	1 (V)
1 (V)	0 (F)	1 (V)
0 (F)	1 (V)	1 (V)
0 (F)	0 (F)	0 (F)

OPERADOR OR

Ejemplo N°1:

Si $a=10$, $b=20$, $c=30$ entonces el valor de verdad de la siguiente expresión es:

$$(a < b) \text{ AND } (b > c)$$

$$(10 < 20) \text{ AND } (20 > 30)$$

$$V \quad \text{AND} \quad F$$

F

Ejemplo N°2:

1. Si $a = 10$, $b = 12$, $c = 13$, $d = 10$

a) $((a > b) \text{ OR } (a < c)) \text{ AND } ((a = c) \text{ OR } (a > = b))$

$$\left(\begin{matrix} F \\ V \end{matrix} \text{ OR } \begin{matrix} V \\ V \end{matrix} \right) \text{ AND } \left(\begin{matrix} F \\ F \end{matrix} \text{ OR } \begin{matrix} F \\ F \end{matrix} \right)$$

$$V \quad \text{AND} \quad F$$

F

b) $((a > = b) \text{ OR } (a < d)) \text{ AND } ((a > = d) \text{ AND } (c > d))$

$$\left(\begin{matrix} F \\ F \end{matrix} \text{ OR } \begin{matrix} F \\ F \end{matrix} \right) \text{ AND } \left(\begin{matrix} V \\ V \end{matrix} \text{ AND } \begin{matrix} F \\ F \end{matrix} \right)$$

$$F \quad \text{AND} \quad F$$

F

c) $\text{NOT } (a = c) \text{ AND } (c > b)$

$$\text{NOT} \left(\begin{matrix} F \\ V \end{matrix} \right) \text{ AND } \left(\begin{matrix} V \\ V \end{matrix} \right)$$

$$V \quad \text{AND} \quad V$$

V

Ejemplo N°3

Supongamos que a Ud. le solicitan evaluar si un número cualquiera se encuentra en el intervalo $[2, 5]$. Su respuesta debe ser en término de verdadero o falso.

Previo

El intervalo matemático $[2, 5]$, se interpreta de la siguiente forma, “son todos los número que se encuentran entre 2 y 5 considerando el 2 y excluyendo al 5”.

Cuando el corchete encierra al número $[2$ indica que se considera al número por tal motivo utilizamos el signo $=$ en la expresión siguiente. Cuando el paréntesis en corchete se utiliza hacia fuera del número o no lo encierra como el caso de $5[$, indica que no se considera su extremo por tal razón no se utiliza el símbolo $=$.

Solución:

Luego, el intervalo en forma lógica es:

$$(x \geq 2) \text{ and } (x < 5)$$

Supongamos que el valor de x es 3, la respuesta debe ser: verdadero.

Hagamos la evaluación.

$$\begin{array}{ccc} (3 \geq 2) & \text{and} & (3 < 5) \\ \mathbf{V} & & \mathbf{V} \\ & & \mathbf{V} \end{array}$$

Si x toma el valor 13 que se encuentra fuera del intervalo, la respuesta debe ser: falso.

Hagamos la evaluación.

$$\begin{array}{ccc} (13 \geq 2) & \text{and} & (13 < 5) \\ \mathbf{V} & \text{and} & \mathbf{F} \\ & & \mathbf{F} \end{array}$$

Observe los siguientes ejemplos:

- a) $]12,56[$ $= (x > 12) \text{ and } (x < 56)$
- b) $] -\infty,56]$ and $[150, \infty+[$ $= (x \leq 56) \text{ and } (x \geq 150)$
- c) $[25,89]$ or $]800, \infty+[$ $= (x \geq 25) \text{ and } (x \leq 89) \text{ or } (x > 800)$

Generalidades de los operadores

1. La expresión $x \geq y$ es igual a la expresión $(x > y) \text{ or } (x = y)$
2. Paréntesis. Se utilizan para anidar expresiones.

Ejemplo:

Expresión	Resultado
$(9 > 3) \text{ or } (5 = 3)$	verdadero
$(9 > 4) \text{ and } (5 < 4)$	verdadero
$(16 = (4 * 4)) \text{ and } 7 < 7$	falso
$((2 + 3) * (4 - 9)) / 5$	-5

3. El orden de prioridad de los operadores es:
 - paréntesis, comenzando por el más interno
 - potencias
 - productos y divisiones

- sumas y restas
- concatenación
- relacionales
- lógicos

4. Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

2.5.6 Tipo de Expresiones

Las expresiones son la representación de un cálculo necesario para la obtención de un resultado. En informática existen tres tipos de expresiones:

1. **Numéricas:** Generan un resultado numérico y se construyen mediante operadores aritméticos:

Ejemplo: $(X+Y) * Z * 4.15 + X$

2. **Alfanuméricas:** Resultados alfanuméricos, se construyen mediante operadores alfanuméricos:

Ejemplo:

$N := \text{"Juan"}$

$\text{"Don "} + N = \text{"Don Juan"}$

3. **Booleanos:** Resultados verdaderos o falso, se construyen mediante operadores relacionales y lógicos:

Ejemplo: $A > 3$ y $B < 5$.

Ejercicios propuestos (OBJETIVO 2.5)

1. Indicar si son ciertas o falsas las expresiones:

- a) "12" + "12" = "24"
- b) "12" + "12" = "1212"
- c) "hola" = "hola"
- d) "hola" = "hola "
- e) $12 + 12 = 24$
- f) verdadero and verdadero
- g) not falso
- h) (verdadero and verdadero) or falso
- i) (falso or falso) and falso
- j) not (verdadero and falso)

2. Realice el siguiente cálculo respetando las prioridades

- a) $4 + 2 * 5$
- b) $23 * 2 / 5$
- c) $46 / 5$
- d) $3 + 5 * (10 - (2 + 4))$
- e) $3 + 5 * (10 - 6)$
- f) $3.5 + 5.09 - 140 / 40$
- g) $3.5 + 5.09 - 3.5$
- h) $2.1 * (1.5 + 3.0 * 4.1)$

3. Transforme las siguientes expresiones algebraicas en expresiones algorítmicas utilizando los operadores informáticos:

a) $M + \frac{N}{P - Q}$

b) $\frac{M}{N} + 4$

c) $\frac{M + N}{P - Q}$

d) $\frac{P + \frac{N}{P}}{Q - \frac{R}{5}}$

e) $\frac{3x - y}{z} + \frac{2xy^2}{z - 1} + \frac{x}{y}$

f)
$$\frac{\frac{a}{b - \frac{c}{d - \frac{e}{f - g}}} + \frac{h+i}{j+k}}$$

4. Resuelva las siguientes concatenaciones:

- a) "12" + "12"
- b) "El resultado " + "es"
- c) "Método"+"lógico"
- d) "123"+"24"+"A"

5. Si $a = 10$, $b = 20$, $c = 30$, entonces

- a) $(a + b) > c$
- b) $(a - b) < c$
- c) $(a - b) = c$
- d) $(a * b) < c$

6. Calcular el valor de las siguientes expresiones aritméticas:

- a) $21 \bmod 7$
- b) $9/2 + 15 \bmod 2$
- c) $(3+6)/2 + 1.5$
- d) $32/3^2$
- e) $2*3 + 5*6/2*3$
- f) $(25-7*3)^{3/4} * 5$
- g) $10 + 38 / (14 - (10 - 12 / (2*3)))$

7. Calcular el resultado de las siguientes expresiones lógicas:

- a) $\text{not } (5 > 6) \text{ and } 7 \leq 4$
- b) $7 > 4 \text{ and } 5 \leq 5 \text{ or } 4 = 5$
- c) $\text{not } (7 = 7) \text{ and } (7 \geq 8 \text{ or } 8 = 6)$
- d) $(5 + 2) \leq 5 \text{ and } 3 * 2 = 5 \text{ or } 7 \leq 2 * 2 \text{ or } 2 * 2 \leq (2 + 2)$
- e) $(\text{not } (14/2 > 8) \text{ or } 5 > 5) \text{ and } (5 \leq 27/3 \text{ or } 5 + 3 \leq 3/2)$
- f) $(3 + 5 * 2) = 12/3 \text{ and } ((5 + 3) = 18/9 \text{ or } 10/2 \leq 9) \text{ or } \text{not } (9 \geq 2)$

8. Expresar el resultado de las siguientes expresiones:

- a) "PEDRO " + "GOMEZ "
- b) "GARCIA " + " - GONZALEZ "
- c) "CALLE- " + "-MAYOR "
- d) "12.465"+"450"+"-k"

9. Escriba los siguientes intervalos numéricos en sus correspondientes intervalos lógicos:

- a) $[5, 15]$ b) $]120, 200]$ c) $[0, 50[$ d) $]15, 30[$ e) $]-\infty, 15]$ and $[30, \infty[$

OBJETIVO 2.6: DESARROLLAR ALGORITMOS QUE INCLUYEN MANEJO DE VARIABLES, ENTRADA, SALIDA Y ESTRUCTURAS DE CONTROL

2.6.1 CONCEPTO DE INSTRUCCIONES

Instrucciones. Son las órdenes que conforman un algoritmo.

2.6.2 CLASIFICACIÓN DE LAS INSTRUCCIONES

2.6.2.1 Instrucciones de entrada

Son instrucciones que se utilizan para tomar datos desde la entrada al sistema, por ejemplo, el teclado y que se guardaran en variables.

Asignación: Permite ingresar un valor a una variable para realizar procesos.

2.6.2.2 Instrucciones de salida

Sirven para presentar en pantalla o en impresora comentarios, constantes, contenidos de las variables y resultados de expresiones.

2.6.2.3 Instrucciones de control de decisión

Las instrucciones condicionales o tomas de decisión permiten realizar acciones alternativas. Significa que la ejecución de una línea o grupos de líneas del programa depende de sí se cumple o no alguna condición.

Para mostrar este tipo de instrucción vamos a incorporar otra forma de solucionar problemas:

Pseudocódigo o pseudolenguaje: describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de lenguaje de programación y palabras claves.

2.6.3 Estructura de los algoritmos en Pseudolenguaje

Normalmente se dividen en tres partes:

1. **inicial**, en ella:

- ✓ se declaran e inicializan las variables que se usarán.
- ✓ se abren archivos
- ✓ se introducen los valores de aquellas variables que van fuera de los ciclos
- ✓ otros, como mensajes.

2. **cuerpo o proceso.** Corresponde a la parte más importante del algoritmo. Normalmente estas instrucciones se deben cumplir un número determinado de veces o hasta que se cumpla cierta condición.

3. **final:**

- ✓ impresión de resultados finales
- ✓ cierre de archivos
- ✓ otros

2.6.4 Nomenclatura de instrucciones que se utilizarán en Pseudolenguaje

Instrucciones de entrada

Se utilizan para tomar datos de entrada que se guardarán en las variables.

Sintaxis: **Read mensaje, nombre_variable**

Ejemplos:

Read “Ingrese dato”, a

Read “Nombre : “, nom

Instrucciones de salida

Sirven para presentar en pantalla o en impresora comentarios, constantes, contenidos de las variables o resultados de expresiones.

Sintaxis: **Print variable o comentarios.**

Ejemplos:

Print “Líquido a pago:”, a

Print “Nombre : “, nom

Nota: El nombre de las variables se indican sin cremillas para que imprima su contenido.

Por ejemplo:

Print “chao” (las cremillas no se imprimen)

Print a (Se imprime el contenido de la variable a)

Print nom (Se imprime el contenido de la variable nom)

Otro ejemplo

nu := 56

Print nu, (mostrará el 56)

Print 4/2+4 (mostrará 6)

Print 8+2*3 (mostrará 14)

Si a=“hola” b=“y” c=“chao”

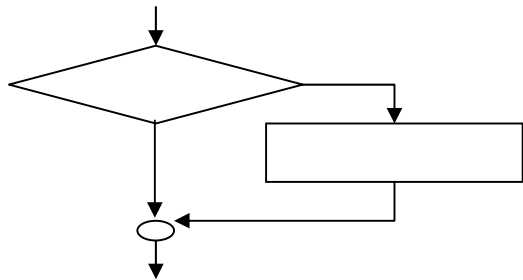
Print a+b+c (Se imprimirá hola y chao).

Comentarios entre líneas

Cuando se desarrollan algoritmos, es muy conveniente documentar ciertas líneas de código indicando que es lo que hacen. Para ello se utiliza el **carácter #**. Veremos su uso en los próximos ejercicios

2.6.5 ESTRUCTURAS DE SELECCIÓN

2.6.5.1 Instrucción de alternativa simple

<p>Alternativa Simple: Controla la ejecución de <i>un conjunto de instrucciones</i> por el cumplimiento o no de una condición de tal forma que, si esta es verdadera se ejecutan, si es falsa no se ejecutan.</p> <p>Sintaxis</p> <p>If (Condición) Then I1; I2; I3!; ...; In End_If</p>	<p>Equivalente en DDF</p> 
--	---

Por ejemplo, en la expresión “si llueve, lleva paraguas”, la acción de llevar paraguas está supeditada a que se cumpla la condición, *si llueve*.

Ejemplo:

Realizar un algoritmo en DDF y pseudolenguaje que permita obtener los cálculos de una venta de computadores. Debe considerar que:

- ✓ Sí el monto de la venta sin IVA es mayor que \$ 300.000 se aplica un descuento del 10% a la venta.
- ✓ Debe calcular el impuesto IVA que es un 18% sobre toda venta.

Entender el problema

Por cada venta hay que ingresar el monto de la venta. Analizar si corresponde descuento. Calcular el IVA e imprimir el total de la venta.

Determinar lo que se intenta hacer

Se quiere el valor total de la venta.

Identificar datos importantes

Valor unitario del computador.

Construir un plan

Ingresar valor unitario del computador en una variable entera.

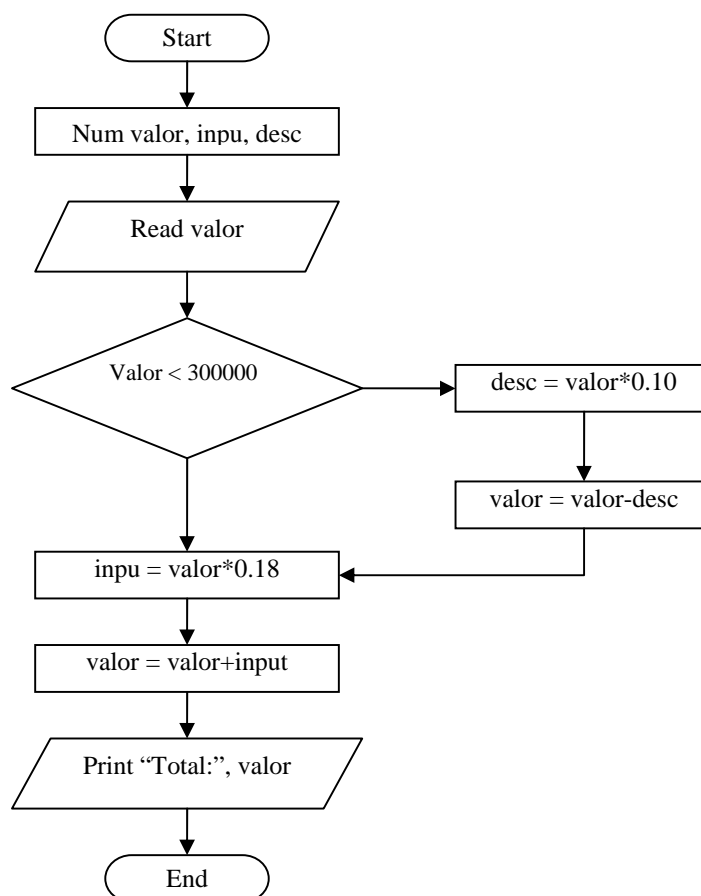
Preguntar si la venta es menor que 300.000. Si es mayor realizar descuento y guardar el resultado

Calcular el IVA y guardar el resultado en una variable numérica

Obtener venta bruta: Sumar venta más IVA y guardar en una variable numérica.

Imprimir venta bruta con un mensaje adecuado.

Solución en DDF



Solución en pseudolenguaje

```

Start
#Se declaran 3 variables
Num Valor, Desc, Inpu
Read Valor
If (Valor > 300000) Then
    Desc := Valor * 0,10
    Valor := Valor - Desc
End_If
Inpu := Valor * 0,18
Valor := Valor + Inpu
Print "Total:", Valor
End
  
```

Verificación del algoritmo

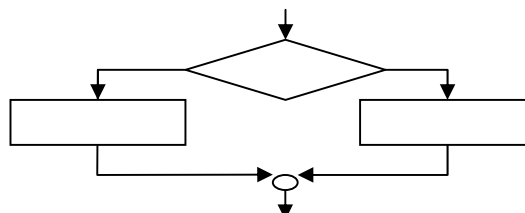
Dados dos valores para computadores: \$350.000 y 290.000, recorra el algoritmo para verificar su funcionamiento.

2.6.5.2 Instrucción de alternativa doble

Alternativa Doble: Controla la ejecución de *Dos Conjuntos De Instrucciones* por el cumplimiento o no de una condición, de tal forma que, si la condición es verdadera, se ejecutan las instrucciones del primer bloque (I1...In), de lo contrario si la condición es falsa, se ejecutan las instrucciones del segundo bloque (J1...Jn).

If (CONDICION) Then
 I1; I2; I3l; ...; In
Else
 J1; J2; J3; ...; Jn
End_If

Equivalente en DDF



A partir de este punto, todos los algoritmos se harán solamente en pseudolenguaje.

Ejemplo:

Realizar un algoritmo en pseudolenguaje que permita obtener los cálculos de una venta de computadores. Debe considerar que:

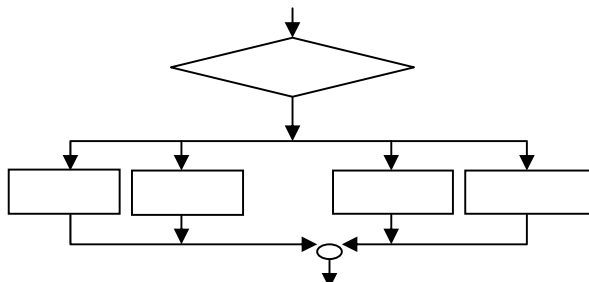
- ✓ Sí el monto de la venta sin IVA es mayor que \$ 300.000 se aplica un descuento del 10% a la venta. Caso contrario se aplicará un descuento del 5%.
- ✓ Debe calcular el impuesto IVA que es un 18% sobre la venta.
- ✓ Debe imprimir el monto del descuento.

Solución

```

Num valor,desc,impu
Print "Ingrese precio: "
Read valor
# Uso de la instrucción de alternativa doble
If (valor > 300000) Then
    desc:= valor * 0,10
Else
    desc:= valor * 0,05
End_If
Print "Descuento: ", desc
Valor := valor – desc
Impu := valor * 0,18
Valor := valor + impu
Print "Total: ", valor
End
  
```

2.6.5.3 Instrucción de alternativa múltiple o anidada

<p>Alternativa Múltiple: Controla la ejecución de varios conjuntos de instrucciones por el valor final de una expresión, de tal forma que cada conjunto de instrucciones está ligado a un posible resultado de la expresión, existiendo además un bloque al final que corresponde a otros posibles valores no definidos.</p> <p>Case Expresión in valor1: Instrucciones break valor2: Instrucciones break valor3: Instrucciones break valorN: Instrucciones break valorOtro: Instrucciones End_Case</p>	<p>Equivalente en DDF</p> 
---	---

La selección de una opción se puede hacer mediante la instrucción **If**. Sin embargo, cuando existe un nido de **If** es complicado seguir el código. La instrucción **Case** es más rápida y más sencilla de manejar. La expresión se evalúa solamente una vez.

Ejemplos de uso de la sentencia condicional.

1. Algoritmo que permita introducir un número por teclado y que envíe un mensaje indicando si es negativo o positivo.

Análisis: *Un número es positivo o negativo, pero nunca ambos. Al cero se le considera positivo, para este ejemplo.*

Solución:

Start

```
#se declara una variable numérica
num j
#uso de la instrucción Read con mensaje incorporado
Read "Introducir un número:", j
# uso de la Instrucción de alternativa doble
If (j >= 0) Then
    Print "es positivo"
Else
    Print "es negativo"
End_If
```

End

2. Desarrollar un algoritmo que permita introducir un número por teclado y que indique si es par o impar.

Análisis: para saber si un número es par o impar hacemos: (número mod 2). Si el resto es 0 entonces es par, caso contrario, impar.

Solución:

```
Start
  num j
  Read "Introducir un número:", j
  # uso de la Instrucción de alternativa doble
  If (j mod 2 = 0) Then
    Print "El numero ", j, "es par"
  Else
    Print "El numero ", j, "es impar"
  End_If
End
```

3. Desarrollar algoritmo que muestre un menú con 4 operaciones aritméticas de la siguiente forma: (no hay validación)

```
Menú Principal
1.- suma
2.- resta
3.- multiplicación
4.- división
```

Solución

```
Start
  #se declaran e inicializan 4 variables
  num a:=0, b:=0, op:=0
  Print "menú principal"
  Print "1.- suma "
  Print "2.- resta "
  Print "3.- multiplicación "
  Print "4.- división "
  Read "Elija opción : ", op
  Read " Ingrese número ", a
  Read " Ingrese otro número ", b
  # Instrucción de alternativa múltiple o anidada
  Case op in
    1:Print " La suma es ", a+b
      break
    2:Print " La resta es ", a-b
      break
    3:Print " La multiplicación es ", a * b
      break
    4:Print " La división es ", a / b
  End_case
End
```

4. Introducir dos números cualquiera. Sumarlos si ambos son mayores que cero. Multiplicarlos si son menores a cero. Sacar la diferencia si uno es positivo y el otro negativo. Indicar con un mensaje si ambos números son iguales a cero.

Solución:

```

Start
  num a, b, c :=0
  Read "Ingrese N°", a
  Read "Ingrese otro N°", b
  #Utilizaremos nido de if
  If (a>0 and b>0) Then
    Print "El resultado es: ", a+b
  Else
    If (a<0 and b<0) Then
      Print "el resultado es :", a*b
    Else
      If (a=0 and b=0) Then
        Print "el resultado es cero"
      Else
        Print "el resultado es :", a-b
      End_if
    End_if
  End_If
End

```

5. Introducir dos números cualquiera. Sumarlos si ambos son mayores que cero. Multiplicarlos si son menores a cero. Sacar la diferencia si uno es positivo y el otro negativo. Indicar con un mensaje si es igual a cero.

Solución:

```

Start
  # Esta solución considera If's No anidados
  num a, b, c :=0
  Read "Ingrese n°", a
  Read "ingrese otro n°", b
  If (a>0 and b>0) Then
    Print "el resultado es :", a+b
  End_If
  If (a<0 and b<0) Then
    Print "el resultado es :", a*b
  End_If
  If (a=0 and b=0) Then
    Print "el resultado es cero"
  End_If
  If (a >0 and b<0 OR a<0 and b>0) Then
    Print "el resultado es :", a-b
  End_If
End

```

EJERCICIOS PROPUESTOS (OBJETIVO 2.6.5)

1. Suponga que un individuo desea invertir su capital en un banco y desea saber cuanto dinero ganará después de un mes si el banco paga a razón de 2% mensual.
2. Ingresar dos números y determinar cual es el mayor o si son iguales.
3. Leer tres números diferentes e imprimir el número mayor de los tres.
4. Ingresar tres números y mostrarlos en forma decreciente
5. Leer 2 números; si son iguales que los multiplique, si el primero es mayor que el segundo que los reste y si no que los sume.
6. Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuanto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.
7. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuanto deberá pagar finalmente por su compra.
8. Un alumno desea saber cual será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:
55% del promedio de sus tres calificaciones parciales.
30% de la calificación del examen final.
15% de la calificación de un trabajo final.
9. Un maestro desea saber que porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.
10. Un hombre desea saber cuanto dinero se genera por concepto de intereses sobre la cantidad que tiene en inversión en el banco. El decidirá reinvertir los intereses siempre y cuando estos excedan a \$7000, y en ese caso desea saber cuanto dinero tendrá finalmente en su cuenta.
11. Determinar si un alumno aprueba o reprueba un curso, sabiendo que aprobará si su promedio de tres calificaciones es mayor o igual a 4,5; reprueba en caso contrario.
12. En un almacén se hace un 20% de descuento a los clientes cuya compra supere los \$1000. ¿Cuál será la cantidad que pagara una persona por su compra?
13. Un obrero necesita calcular su salario semanal, el cual se obtiene de la siguiente manera:
 - ✓ Si trabaja 40 horas o menos se le paga \$16 por hora
 - ✓ Si trabaja más de 40 horas se le paga \$16 por cada una de las primeras 40 horas y \$20 por cada hora extra.

14. Para el intervalo cerrado $[347, 2342]$, desarrollar un algoritmo que imprima, cuente y sume el cuadrado de los múltiplos de 7 y que además, cuente los múltiplos de 3 que se encuentren en dicho intervalo.
15. En una empresa de n trabajadores se ha decidido reajustar el sueldo a todo el personal en forma escalonada. Hacer un algoritmo que permita ingresar la edad y el sueldo actual de un trabajador para que entregue el cálculo del nuevo sueldo de acuerdo a la siguiente tabla:

Si es menor de 26 años: aumento del 10%
 Entre 26 y 35 años: aumento del 20%
 Entre 36 y 50 años: aumento del 50%
 Sobre 50 años: aumento del 70%

2.6.6: INSTRUCCIONES DE CONTROL DE REPETICIÓN

Concepto de Bucle o ciclo

En informática, la mayoría de las veces las tareas que realiza el computador son repetitivas, lo único que varía son los valores de los datos con los que se está operando. Se llama bucle o ciclo a todo proceso que se repite un número de veces dentro de un programa.

Contador: Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción.

Los contadores normalmente se utilizan en las estructuras de repetición con la finalidad de contar sucesos o acciones internas de bucle o ciclo.

Se debe realizar una operación de inicialización que consiste en asignar al contador un valor inicial. Se situará antes y fuera del ciclo.

Sintaxis:

Nombre_del_contador:=valor_inicial

Los incrementos o decrementos del contador se realizan de la siguiente forma:

Sintaxis:

Nombre_del_contador := Nombre_del_contador + valor_constante

Nombre_del_contador := Nombre_del_contador - valor_constante

Primero se evalúa la expresión de la derecha del signo :=, realiza la suma o la resta y su resultado se lo asigna a lo que hay a la izquierda del signo :=

Ejemplos:

- ✓ `cont := cont + 1`
- ✓ `a := a - 3`
- ✓ `a := a + 2`
- ✓ `con := con + 5`
- ✓ `m := 7`
- ✓ `m := m + 7`
- ✓ `Print m` (mostrará 14)

Acumulador: Son variables cuyo valor se incrementa o decrementa en una cantidad variable. Al igual que los contadores también necesitan inicializarse fuera del ciclo.

Sintaxis:

Nombre_del_acumulador:=valor_inicial

Su operación dentro del bucle queda definida en la siguiente expresión:

Sintaxis:

Nombre_del_acumulador:= Nombre_del_acumulador + nombre_variable

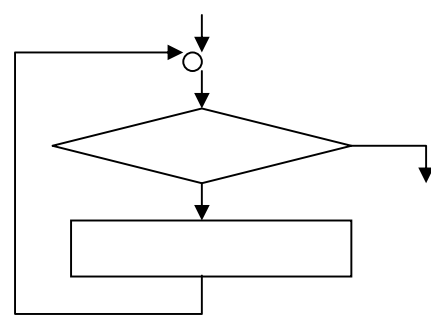
Ejemplo:

- ✓ `saldo := saldo + deposito`
- ✓ `saldo := saldo - retiro`

Concepto de Interruptor: Un interruptor o bandera (switch) es una variable que puede tomar los valores 1(verdadero) ó 0 (falso) a lo largo de la ejecución de un programa, dando así información de una parte a otra del mismo. Puede ser utilizado para control de ciclo.

2.6.7: FORMAS COMUNES DE ESTRUCTURAS DE REPETICIÓN

2.6.7.1. Instrucción While ...End While

<p>While : Repite la ejecución de un conjunto de instrucciones, de tal forma que estas se ejecutan mientras la condición sea verdadera, la condición, será evaluada siempre antes de cada repetición.</p> <p>Sintaxis:</p> <p>While (condición) Instrucciones End_While</p>	<p>Equivalente en DDF</p> 
---	---

Por ejemplo:

```
num a :=0
While (a < 20)
    Print a
    a := a + 1
End_While
```

Lo primero que hace el algoritmo es examinar la condición:

- ✓ Si se cumple: realiza todas las instrucciones que se encuentran dentro del ciclo y lo hará hasta que se deje de cumplir la condición, entonces sale del ciclo.
- ✓ Si no se cumple: no entrará al ciclo por lo tanto, no ejecuta ninguna instrucción que se encuentre dentro del ciclo, es decir, se lo saltará.

La condición del bucle no tiene por que ser única. Puede haber más de una y estar unidas por operadores lógicos

Por ejemplo:

```
While (a < 20 and resp = "s")
    .....
    instrucciones
    .....
End_While
```


Ejemplos de uso de la sentencia de ciclos.

1. Hacer un algoritmo que imprima los números naturales del 1 al 100 en orden ascendente.

Análisis: Se requiere una variable que vaya tomando los valores desde 1 hasta 100, de uno en uno. Como debemos partir de 1 debemos inicializar la variable en 1.

Se debe utilizar un ciclo para que imprima el valor de la variable y al mismo tiempo vaya incrementándola 1 hasta llegar al 100.

Variable numérica cont se utilizará como contador.

Solución

```
Start
  num cont := 1
  While (cont <= 100)
    Print cont
    cont := cont + 1
  End_While
End
```

2. Hacer un algoritmo que imprima los números naturales del 100 al 15 en orden decreciente.

Análisis: Se requiere una variable que vaya tomando los valores desde 100 hasta 15, de uno en uno. Como debemos partir de 100 debemos inicializar la variable en 100.

Se debe utilizar un ciclo para que imprima el valor de la variable y al mismo tiempo vaya decrementándola en 1 hasta llegar a 15.

Variable numérica cont se utilizará como contador.

Solución

```
Start
  num cont := 100
  While (cont >= 15)
    Print cont
    cont := cont - 1
  End_While
End
```

3. Hacer un algoritmo que imprima la suma de los 100 primeros números naturales.

Análisis: Aquí el trabajo es doble; por un lado se debe generar los números y por otro, a medida que se generan, se suman.

Para la primera operación se necesita un contador que aumente de 1 en 1 y para la segunda, un acumulador que acumule la suma de los números generados.

La impresión de los números se realiza fuera del bucle pues solamente se debe imprimir el resultado final.

Variables: c = contador para generar los números.

acu = acumulador donde se va realizando la suma de los números.

Solución

```

Start
  #El acumulador se debe inicializar en cero
  num c := 1, acu := 0
  While (c <= 100)
    acu := acu + c
    c := c + 1
    # verificar que sucede si el acumulador
    # se pone después del contador
  End_While
  Print " La suma es ", acu
End

```

4. Hacer un algoritmo que imprima todos los números naturales que hay desde 1 hasta un número que se introduce por teclado.

Análisis: Hay que introducir un número por teclado, para ello necesitamos una variable. Se deben imprimir los números desde 1 hasta dicho número.

El proceso de introducción del número por teclado debe ser antes del ciclo para saber hasta que número debemos imprimir.

Variables: c = contador para generar los números.
nn = Para recibir el número que se ingresa por teclado.

Solución

```

Start
  num nn, c:=1
  Read "Ingrese un número", nn
  While (c <= nn)
    Print c
    c := c + 1
  End_While
End

```

5. Escriba un algoritmo que permita obtener la nota final de 15 alumnos y el promedio general del curso. Para la nota final debe considerar lo siguiente:

- ✓ El promedio de las notas parciales equivale al 60% de la nota final.
- ✓ El examen equivale al 40 % de la nota final.

Análisis: Debe ingresar 15 veces el promedio de las notas parciales del semestre y 15 veces la nota del examen. Cada vez que se obtenga el promedio final de un alumno, se acumula para obtener el promedio general del curso.

Variables:

Contador que permita controlar que el proceso se repita 15 veces.
Variable numérica para ingresar el promedio del semestre
Variable numérica para ingresar la nota del examen.
Variable numérica para acumular el promedio general del curso.

Solución

Start

```
Num cont, prom, acuProm, exaNot, finalAlu
cont := 1
acuProm:=0
While ( cont <= 15 )
    Print "Ingrese Promedio del Alumno: ", cont
    Read prom
    Read "Ingrese Nota Examen: ", exaNot
    finalAlu := (prom * 0.6) + (exaNot * 0.4)
    acuProm:=acuProm + finalAlu
    Print "La nota final del alumno ", cont, "es ", finalAlu
    cont := cont + 1
```

End_While

```
Print "El promedio general del curso es", acuProm/15
```

End

Obs: Esta solución utiliza un ciclo While determinado, es decir, se conoce a priori el número de iteraciones, en este caso 15. Si en número de alumnos fuera indeterminado, se pueden controlar las iteraciones interactivamente con el usuario, por ejemplo, a través de una variable de tipo carácter, la cual debe ser leída y dependiendo de su valor de verdad ("S" o "N"), se realizara o no una nueva iteración. Para esta solución el ciclo estará controlado por esta variable. Además realizaremos la validación de la variable de tal forma que solamente pueda tomar los valores "S" o "N".

Start

```
# Solución que utiliza el ciclo While indeterminado
Num cont, prom, acuProm, exaNot, finalAlu
# Se declara e inicializa la variable Resp
Char Resp := "S"
cont := 0
acuProm:=0
While ( Resp = "S" )
    cont := cont + 1
    Read "Ingrese promedio : ", prom
    Read "Ingrese Nota Examen: ", exaNot
    finalAlu := (prom * 0.6) + (exaNot * 0.4)
    acuProm:=acuProm + finalAlu
    Print "La nota final del alumno es: ", finalAlu
    cont := cont + 1
    Print "Desea otro ingreso S/N :"
```

Read Resp

```
While ( Resp <> "N" or Resp <> "S" )
```

```
    Read Resp`
```

End_While

End_While

```
Print "El promedio general del curso es", acuProm / cont
```

End

6. Hacer algoritmo que permita imprimir 20 veces en forma alternadamente: hola y chao. Utilizar el concepto de interruptor.

Solución

Start

```
#la variable sw se usará como switch
num sw := 0, i := 1
While (i<=20)
    If (sw = 0) Then
        Print "hola"
        #imprime y cambia el valor de switch
        sw:=1
    Else
        Print "chao"
        #imprime y cambia el valor de switch
        sw :=0
    End_If
    i := i + 1
End_While
```

End

7. Hacer un algoritmo que cuente en forma independiente los pares e impares mediante switch del intervalo [1,1000].

Solución

Start

```
#la variable sw se usará como switch
num sw := 0, i:=1, sump:=0, sumip:=0
While (i<=1000)
    If (sw = 0) Then
        sumip := sumip + 1
        #imprime y cambia el valor de switch
        sw:=1
    Else
        sump := sump + 1
        # cambia el valor de switch
        sw :=0
    End If
    i := i + 1
End_While
Print "La cantidad de número pares es: ", sump
Print "La cantidad de número impares es: ", sumip
```

End

8. Introducir dos números por teclado. Imprimir los números naturales que hay entre ambos números empezando por el más pequeño. Contar cuantos hay. Contar cuantos son pares. Calcular la suma de los impares.

Solución

Start

```

num nuA, nuB , cont := 0, par := 0, sumimp := 0
Read "Ingrese primer número: ", nuA
Read "Ingrese segundo número: ", nuB
#para verificar quien es el más pequeño, nos
#aseguramos que será nuA
If (nuA > nuB) Then
    aux := nuA
    nuA := nuB
    nuB := aux
End_If
While (nuA <= nuB)
    Print nuA
    cont := cont + 1
    If (nuA MOD 2 = 0) Then
        par:= par + 1
    Else
        sumimp := sumimp + nuA
    End_If
    nuA := nuA + 1
End_While
Print "Cantidad de números entre ambos :", cont
Print "Cantidad de pares :", par
Print "La suma de los impares ", sumimp

```

End

2.6.7.2. Instrucción For

Cuando se conoce de antemano los límites en que varía una variable, es posible utilizar otra estructura de ciclo.

Instrucción For

For: Repite un grupo de instrucciones un número específico de veces.

Sintaxis:

```

For (Valor_inicial, condición, incremento)
    sentencias
End_For

```

Descripción de la instrucción

Valor_inicial, es una sentencia de asignación que sirve como variable de control del bucle o ciclo.

Condición, es una expresión que indica la condición que se debe cumplir para que continúe el bucle o ciclo.

Incremento, define como va cambiando el valor de la variable de control cada vez que se repite el ciclo.

Sentencias, conjunto de instrucciones que se van a repetir tantas veces como dure el ciclo.

Ejemplo N°1: Algoritmo que muestra los números enteros entre 1 y 100.

```

Start
    num i
    For (i:=1, i<=100, i:=i+1)
        Print i
    End_For
End

```

2.6.7.3. Instrucción Do....While (Condición)

Do...While (Condición): Repite un grupo de instrucciones de tal forma que estas se ejecutan mientras la condición sea **Verdadera**. La condición será evaluada siempre después de cada repetición. El conjunto de instrucciones se ejecuta al menos una vez.

Sintaxis

```

Do
    Instrucciones
While (Condición)

```

Ejemplo

Construya un algoritmo en pseudolenguaje que permita a un usuario imprimir boletos de avión desde Santiago hacia el Norte del país.

Por cada boleto se ingresará:

- ✓ Nombre del pasajero.
- ✓ Destino del vuelo (1=Primera Región 5=Quinta Región).
- ✓ Clase (E:ejecutiva, T:turista).

Los siguientes son los valores de los vuelos para la clase turista, según región de destino la clase ejecutiva lleva un recargo de \$20.000 en todos los destinos.

Región	Turista
01	\$100.000
02	\$80.000
03	\$60.000
04	\$50.000
05	\$30.000

Se requiere que el programa imprima un boleto con el nombre del pasajero, la región a la que viaja y el precio a pagar por el vuelo.

Utilizaremos la instrucción Do..While para validar algunos datos.

Solución**Start**

```
num Coddex, Precio
char NomCli (20), Región(18), clase (1)
Print  "Ingrese Nombre Cliente"
Read NomCli
```

Do

```
    Print  "Ingrese Código De Destino"
    Read Coddex
```

```
While (Coddex < 1 or Coddex > 5)
```

Do

```
    Print  "Ingrese Clase E:ejecutiva, T:turista"
    Read Clase
```

```
While (Clase <> "E" and Clase <> "T")
```

Case Coddex in

```
    1:Región:= "Primera Región"
```

```
        Precio:=100000
```

```
        Break
```

```
    2:Región:= "Segunda Región"
```

```
        Precio:=80000
```

```
        Break
```

```
    3:Región:= "Tercera Región"
```

```
        Precio:=60000
```

```
        Break
```

```
    4:Región:= "Cuarta Región"
```

```
        Precio:=40000
```

```
        Break
```

```
    5:Región:= "Quinta Región"
```

```
        Precio:=30000
```

End_Case

```
If (Clase= "E") Then
```

```
    Precio:=Precio + 20000
```

End_If

```
Print  "Nombre", NomCli
```

```
Print  "Región Destino", Región
```

```
Print  "Valor Pasaje", Precio
```

End

EJERCICIOS PROPUESTOS DE CICLOS (OBJETIVO 2.6.6)

1. Calcular el promedio de un alumno que tiene 7 notas en la asignatura “Programación II”.
2. Leer 10 números e imprimir solamente los números positivos
3. Leer 20 números e imprimir cuantos son positivos, cuantos negativos y cuantos son ceros.
4. Leer 15 números negativos, convertirlos a positivos e imprimir dichos números.
5. Suponga que se tiene un conjunto de calificaciones de un grupo de 40 alumnos. Realizar un algoritmo para calcular la calificación media y la calificación más baja de todo el grupo.
6. Calcular e imprimir la tabla de multiplicar de un número cualquiera. Imprimir el multiplicando, el multiplicador y el producto.
7. Simular el comportamiento de un reloj digital, imprimiendo la hora, minutos y segundos de un día desde las 0:00:00 horas hasta las 23:59:59 horas
8. Una compañía de seguros tiene contratados a N vendedores. Cada uno hace tres ventas a la semana. Su política de pagos es que un vendedor recibe un sueldo base y un 10% extra por comisiones de sus ventas. El gerente de la compañía desea saber cuanto dinero obtendrá en la semana cada vendedor por concepto de comisiones por las tres ventas realizadas y cuanto le corresponderá tomando en cuenta su sueldo base y sus comisiones.
9. En una empresa se requiere calcular el salario semanal de cada uno de los n obreros que laboran en ella. El salario se obtiene de la siguiente forma:
 - ✓ Si el obrero trabaja 40 horas o menos se le paga \$20 por hora
 - ✓ Si trabaja más de 40 horas se le paga \$20 por cada una de las primeras 40 horas y \$25 por cada hora extra.
10. Diseñe un algoritmo que determine cuantos hombres y cuantas mujeres se encuentran en un grupo de N personas. Suponga que los datos son extraídos persona por persona.
11. Obtener el promedio de calificaciones de un grupo de n alumnos.
12. Una persona desea invertir su dinero en un banco, el cual le otorga un 2% de interés mensual. ¿Cuál será la cantidad de dinero que esta persona tendrá al cabo de un año si la ganancia de cada mes es reinvertida?.
13. Calcular el promedio de edades de hombres, mujeres y de todo un grupo de alumnos.
14. Encontrar el menor y el mayor valor de un conjunto de n números dados.

15. En un supermercado un cajero captura los precios de los artículos que los clientes compran e indica a cada cliente cual es el monto de lo que deben pagar. Al final del día le indica a su supervisor cuanto fue lo que cobro en total a todos los clientes que pasaron por su caja.
16. Cinco miembros de un club contra la obesidad desean saber cuanto han bajado o subido de peso desde la última vez que se reunieron. Para esto se debe realizar un ritual de pesaje en donde cada uno se pesa en diez básculas distintas para así tener el promedio más exacto de su peso. Si existe diferencia positiva entre este promedio de peso y el peso de la última vez que se reunieron, significa que subieron de peso. Pero si la diferencia es negativa, significa que bajaron. Lo que el problema requiere es que por cada persona se imprima un letrero que diga: "SUBIO" o "BAJO" y la cantidad de kilos que subió o bajo de peso.
17. Se desea obtener el promedio de g grupos que están en un mismo año escolar; siendo que cada grupo puede tener n alumnos que cada alumno puede llevar m materias y que en todas las materias se promedian tres calificaciones para obtener el promedio de la materia. Se desea desplegar el promedio de los grupos, el promedio de cada grupo y el promedio de cada alumno.
18. En una tienda de descuento las personas que van a pagar el importe de su compra llegan a la caja y sacan una bolita de color, que les dirá que descuento tendrán sobre el total de su compra. Determinar la cantidad que pagará cada cliente desde que la tienda abre hasta que cierra. Se sabe que si el color de la bolita es roja el cliente obtendrá un 40% de descuento; si es amarilla un 25% y si es blanca no obtendrá descuento.
19. En un supermercado una ama de casa pone en su carrito los artículos que va tomando de los estantes. La señora quiere asegurarse de que el cajero le cobre bien lo que ella ha comprado, por lo que cada vez que toma un artículo anota su precio junto con la cantidad de artículos iguales que ha tomado y determina cuanto dinero gastara en ese artículo; a esto le suma lo que ira gastando en los demás artículos, hasta que decide que ya tomo todo lo que necesitaba. Ayúdale a esta señora a obtener el total de sus compras.
20. Un teatro otorga descuentos según la edad del cliente. Determinar la cantidad de dinero que el teatro deja de percibir por cada una de las categorías. Tomar en cuenta que los niños menores de 5 años no pueden entrar al teatro y que existe un precio único en los asientos. Los descuentos se hacen tomando en cuenta el siguiente cuadro:

Categorías	Edad	Descuento
Categoría 1	5 - 14	35 %
Categoría 2	15 - 19	25 %
Categoría 3	20 - 45	10 %
Categoría 4	46 - 65	25 %
Categoría 5	66 en adelante	35 %

21. La presión, volumen y temperatura de una masa de aire se relacionan por la fórmula:

$$\text{masa} = \frac{\text{presión} * \text{volumen}}{0.37 * (\text{temperatura} + 460)}$$

Calcular el promedio de masa de aire de los neumáticos de n vehículos que están en compostura en un servicio de alineación y balanceo. Los vehículos pueden ser motocicletas o automóviles.

22. Determinar la cantidad semanal de dinero que recibirá cada uno de los n obreros de una empresa. Se sabe que cuando las horas que trabajo un obrero exceden de 40, el resto se convierte en horas extras que se pagan al doble de una hora normal, cuando no exceden de 8; cuando las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se paga por una hora normal y el resto al triple.
23. Una persona que va de compras a la tienda “Enano, S.A.”, decide llevar un control sobre lo que va comprando, para saber la cantidad de dinero que tendrá que pagar al llegar a la caja. La tienda tiene una promoción del 20% de descuento sobre aquellos artículos cuya etiqueta sea roja. Determinar la cantidad de dinero que esta persona deberá pagar.
24. Un censador recopila ciertos datos aplicando encuestas para el último Censo Nacional de Población y Vivienda. Desea obtener de todas las personas que alcance a encuestar en un día, que porcentaje tiene estudios básicos, medios, superiores, y estudios de postgrado.
25. Un jefe de casilla desea determinar cuantas personas de cada una de las secciones que componen su zona asisten el día de las votaciones. Las secciones son: norte, sur y centro. También desea determinar cuál es la sección con mayor número de votantes.
26. Un negocio de copias tiene un límite de producción diaria de 10.000 copias si el tipo de impresión es offset y de 50.000 si el tipo es estándar. Si hay una solicitud nueva, el empleado tiene que verificar que las copias pendientes hasta el momento y las copias solicitadas no excedan del límite de producción. Si el límite de producción se excediera el trabajo solicitado no podría ser aceptado. El empleado necesita llevar un buen control de las copias solicitadas hasta el momento para decidir en forma rápida si los trabajos que se soliciten en el día se aceptan o no.
27. Calcular la suma siguiente:
 $100 + 98 + 96 + 94 + \dots + 0$ en este orden
28. Leer 50 calificaciones de un grupo de alumnos. Calcule y escriba el porcentaje de reprobados. Tomando en cuenta que la calificación mínima aprobatoria es de 70.
29. Leer por cada alumno de Diseño estructurado de algoritmos su número de control y su calificación en cada una de las 5 unidades de la materia. Al final que escriba el número de control del alumno que obtuvo mayor promedio. Suponga que los alumnos tienen diferentes promedios.
30. El profesor de una materia desea conocer la cantidad de sus alumnos que no tienen derecho al examen de nivelación. Para tener derecho, el alumno debe tener un promedio igual o superior a tres en el promedio de las 5 unidades.
Diseñe un algoritmo que lea las calificaciones obtenidos en las 5 unidades por cada uno de los 40 alumnos y escriba la cantidad de ellos que no tienen derecho al examen de nivelación.

31. Diseñe un algoritmo que lea los 250.000 votos otorgados a los 3 candidatos a alcalde e imprima el número del candidato ganador y su cantidad de votos.
32. Suponga que tiene usted una tienda y desea registrar las ventas en su computador. Diseñe un algoritmo que lea por cada cliente, el monto total de su compra. Al final del día que escriba la cantidad total de ventas y el número de clientes atendidos.
33. Suponga que tiene una tienda y desea registrar sus ventas por medio de un computador. Diseñe un pseudocódigo que lea por cada cliente:
 - a) El monto de la venta,
 - b) Calcule e imprima el IVA ,
 - c) Calcule e imprima el total a pagar,
 - d) Lea la cantidad con que paga el cliente,
 - e) Calcule e imprime el cambio.

Al final del día deberá imprimir la cantidad de dinero que debe haber en la caja.

UNIDAD 3: INTRODUCCION AL LENGUAJE C.

OBJETIVO 3.1: REvisa los elementos de sintaxis y estructura de un programa en lenguaje C.

3.1.1: Describe las etapas de un programa en lenguaje C.

El lenguaje C se conoce como un lenguaje compilado. Existen dos tipos de lenguaje: interpretados y compilados. Los interpretados son aquellos que necesitan del código fuente para funcionar y van traduciendo las instrucciones en pequeños grupos al lenguaje de máquina (P.ej: Basic). Los compilados, en cambio, convierten el código fuente en un archivo o fichero llamado objeto, es decir convierten todo el código a lenguaje de máquina. Este es el caso de la mayoría de las versiones del lenguaje C.

Es por ello que la creación de un programa en C debe pasar por las etapas de Edición, compilación y Ejecución.

La etapa de Edición consiste en transcribir el programa a algún editor. (la mayoría de las versiones de C, trae un editor propio), creando así un archivo que contiene el código del programa (fichero fuente). Los editores trabajan de manera similar a un procesador de textos, pero con las funcionalidades básicas, de esta forma el archivo se puede modificar, cortar, copiar y pegar trozos de texto, etc.

Una vez escrito el programa es necesario compilarlo, esto significa que se convierte el programa que se encuentra en lenguaje de alto nivel al lenguaje de máquina, creando así el fichero objeto. Si la compilación no es exitosa se entregará un listado con los posibles errores detectados por el compilador, estos se presentan en una ventana donde se indica por cada mensaje de error el número de línea donde fue detectado y el tipo de error. Por el contrario si la compilación es exitosa se podrá ejecutar en forma inmediata. Otro punto importante es que el proceso de compilación se puede realizar en conjunto con la ejecución, es decir Compilar y ejecutar, o bien por separado, primero se compila y posteriormente se ejecuta, en ambos casos si el programa presenta errores no se podrá ejecutar, hasta solucionar los errores.

3.1.2. REvisa la estructura de un programa computacional en lenguaje C.

Todo programa en C consta de una o más funciones, una de las cuales se llama **main**. El programa comienza en la función main, desde la cual es posible llamar a otras funciones. Las definiciones de las funciones adicionales se deben realizar aparte, bien precediendo o siguiendo a main.

Cada función estará formada por la cabecera de la función, compuesta por el valor de retorno, nombre de la misma y la lista de argumentos (si los hubiese), la declaración de las variables a utilizar y la secuencia de sentencias a ejecutar.

Las instrucciones compuestas se encierran con un par de llaves { }. Las llaves pueden

contener combinaciones de instrucciones elementales (también llamadas instrucciones de expresión) y otras instrucciones compuestas, así se pueden generar instrucciones compuestas anidadas.

Las instrucciones de expresión deben terminar siempre con un punto y coma. (;).

Ejemplo:

declaraciones globales

```
valor_de_retorno main()
{
    variables locales
    bloque
}
```

```
Valor_de_retorno funcion1()
{
    variables locales
    bloque
}
```

Nota: en algunos compiladores, siempre es necesario incluir un valor de retorno, incluso cuando la función no devuelve valor alguno, en ese caso puntual es necesario indicar el tipo de dato vacío, con la palabra **void**.

Es conveniente añadir comentarios (cuantos más mejor) para poder saber que función tiene cada parte del código, en caso de que no lo utilicemos durante algún tiempo. Además facilitaremos el trabajo a otros programadores que puedan utilizar nuestro archivo fuente.

Para poner comentarios en un programa escrito en C usamos los símbolos **/*** y ***/**:

```
/* Este es un ejemplo de comentario */
```

```
/* Un comentario también puede  
estar escrito en varias líneas */
```

El símbolo **/*** se coloca al principio del comentario y el símbolo ***/** al final.

Un identificador es el nombre que se asigna a las variables y funciones. Está formado por una secuencia de letras y dígitos, aunque también acepta el carácter de subrayado **_**. Por contra no acepta los acentos ni la ñ/Ñ.

El primer carácter de un identificador no puede ser un número, es decir que debe ser una letra o el símbolo **_**.

Se diferencian las mayúsculas de las minúsculas, así **num**, **Num** y **nuM** son distintos identificadores.

A continuación se presentan algunos ejemplos de identificadores válidos y no válidos:

Válidos	No válidos
<code>_num</code>	<code>1num</code>
<code>var1</code>	<code>número2</code>
<code>fecha_nac</code>	<code>año_nac</code>

Ejemplo: programa que permite calcular el área de un círculo:

```
/*Programa para calcular el área de un círculo*/
#include <stdio.h>
main()
{
    float radio, area;
    printf("Radio = ? ");
    scanf("%f ", &radio);
    area= 3.14159 * radio *radio;
    printf("Area = %f ", area);
}
```

Nota: Es importante señalar que debido a que estos programas se ejecutan en entorno D.O.S., es necesario incluir una pausa para poder lograr visualizar en la ventana los datos. Esta pausa se puede realizar agregando después del ingreso del valor del radio la instrucción **getchar()**; y luego antes de finalizar el código. (revise el compilador usado, en algunos la función se llama **getch()**, y se encuentra en la librería **conio.h**, en Dev c++, por ejemplo, existe la función **system("pause")**; que permite realizar la pausa.)

Es decir:

```
#include <stdio.h>
main()
{
    float radio, area;
    printf("Radio = ? ");
    scanf("%f ", &radio);
    getchar();
    area= 3.14159 * radio *radio;
    printf("Area = %f ", area);
    getchar();
}
```

Para el resto de los ejemplos no se agregará la pausa, quedando esto sujeto al compilador usado para ejecutar los programas.

3.1.3: EXPLICA EL PROCEDIMIENTO DE COMPILACIÓN DE PROGRAMAS EN LENGUAJE C.

Como se señaló en el punto 3.1.1, los programas en C tienen una serie de etapas, una de ellas es la compilación, esta se debe realizar para poder ejecutar el programa y obtener así un

archivo ejecutable. Dependiendo de la versión de C que se disponga, existen algunas pequeñas variaciones en relación a la interfaz de trabajo, sin embargo son diferencias sutiles y por lo tanto no cambia en forma significativa el proceso.

A continuación se describe el proceso de compilación considerando el Editor del Turbo C++:

La Ventana principal del Turbo C++, presenta una interfaz con barra de menú, donde al seleccionar una de las opciones de ella se desplegará un submenú, con una serie de opciones. Por ejemplo el menú File, contiene opciones para crear un programa nuevo, abrir uno ya existente, guardar, imprimir, etc.

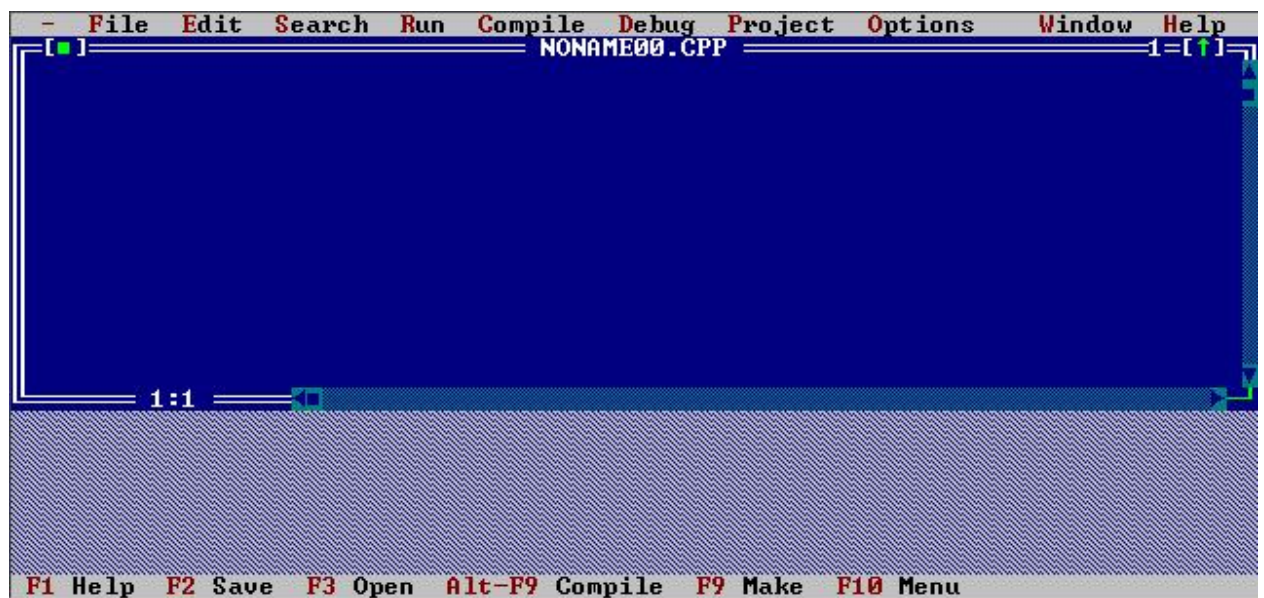


Figura 3.1: Ventana principal de Turbo C++

Debajo de la barra de menú se encuentra el area de trabajo, donde se debe introducir el nuevo programa, o bien editar uno ya existente. Las funciones de edicion propias de un editor de texto se mantienen exactamente igual (copiar, cortar, pegar).

Una vez que el programa se encuentra ingresado y almacenado (es ideal guardarlo antes de compilarlo) se puede compilar, para ello debe ir al menú Compile y elegir la opción compile. Con esto el programa se revisa y si esta correctamente escrito, podrá ejecutarse. Para ejecutar un programa debe ir al menú Debug y elegir la opción Run.

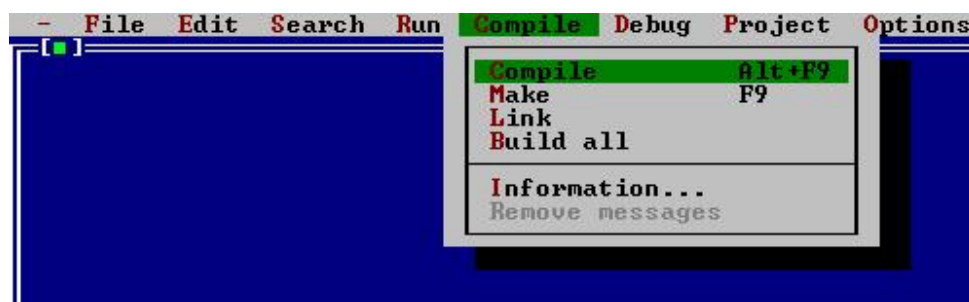


Figura 3.2: Ventana de Turbo C++, Menú Compile

Es importante destacar que la opción Run del menú Run, también realiza el proceso de compilación, es decir, compila y ejecuta.

Si el programa presenta errores, aparece una ventana con un listado de los mensajes de error, estos permiten tener una idea del problema que ocurre y la línea de código donde se detectó. Si el programa presenta muchos errores, es posible que no sean detectados todos en una primera compilación y por lo tanto se deberán ir solucionando por pasos.

Existen herramientas, dentro del editor que permiten depurar el programa, realizar seguimientos, etc. Sin embargo esto no es materia de estudio en esta asignatura, de todas formas se puede fomentar en el alumno la capacidad de autoestudio e investigación para averiguar el funcionamiento y aplicación de estas herramientas.

Otra versión de C, es la Dev C++, en este caso la interfaz de trabajo es gráfica y las opciones son bastante similares a las encontradas en Turbo C.

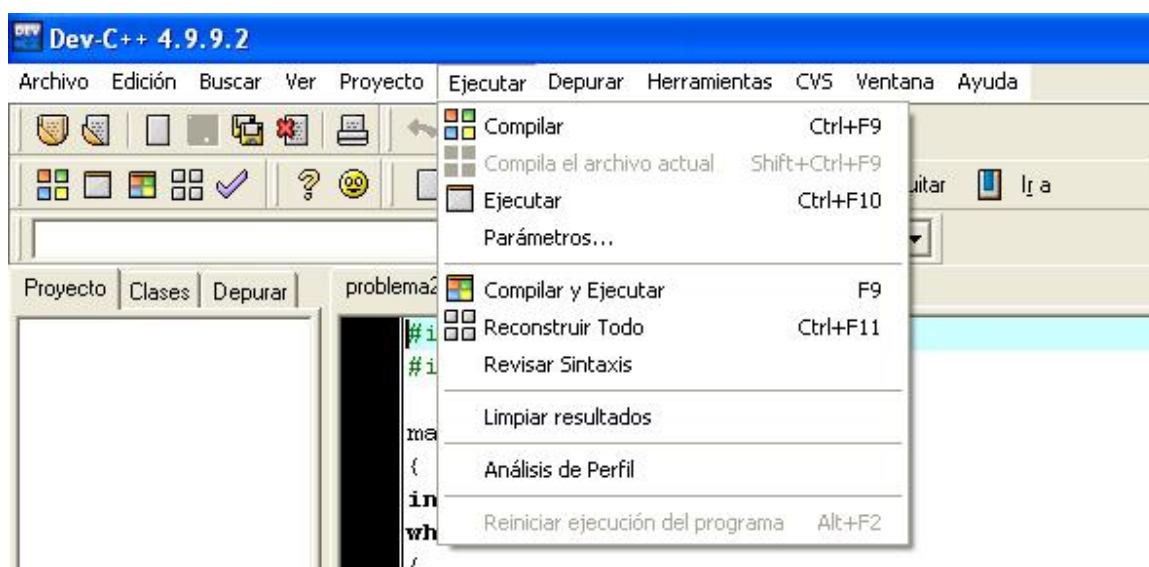


Figura 3.3: Ventana de Dev C++, Menú Ejecutar

OBJETIVO 3.2: TRADUCIR LOS ALGORITMOS ESCRITOS EN PSEUDOLENGUAJE A LENGUAJE C, DETECTANDO Y CORRIGIENDO ERRORES PARA UNA EJECUCIÓN SATISFACTORIA.

3.2.1: EXPLICA LOS TIPOS DE DATOS SOPORTADOS POR EL LENGUAJE C.

En C, existen básicamente cuatro tipos de datos, aunque es posible definir tipos de datos propios a partir de estos cuatro. A continuación se detalla su nombre, el tamaño que ocupa en memoria y el rango de sus posibles valores.

TIPO	Tamaño	Rango de valores
char	1 byte	-128 a 127
int	2 bytes	-32768 a 32767
float	4 bytes	3'4 E-38 a 3'4 E+38
double	8 bytes	1'7 E-308 a 1'7 E+308

Existen además cualificadores de tipo, que permiten ampliar algunos tipos básicos de datos.

- **signed**

Le indica a la variable que va a llevar signo. Es el utilizado por defecto.

	tamaño	rango de valores
signed char	1 byte	-128 a 127
signed int	2 bytes	-32768 a 32767

- **unsigned**

Le indica a la variable que no va a llevar signo (valor absoluto).

	tamaño	rango de valores
unsigned char	1 byte	0 a 255
unsigned int	2 bytes	0 a 65535

- **short**

Rango de valores en formato corto (limitado). Es el utilizado por defecto.

	tamaño	rango de valores
short char	1 byte	-128 a 127
short int	2 bytes	-32768 a 32767

- **long**

Rango de valores en formato largo (ampliado).

	tamaño	rango de valores
long int	4 bytes	-2.147.483.648 a 2.147.483.647
long double	10 bytes	-3'36 E-4932 a 1'18 E+4932

También es posible combinar calificadores entre sí:

signed long int = long int = long

unsigned long int = unsigned long 4 bytes 0 a 4.294.967.295 (El mayor entero permitido en 'C')

Para asociar un tipo de dato a una o más variables, estas deben ser declaradas. Es necesario declarar todas las variables antes de ser utilizadas.

La declaración consta de un tipo de datos, seguido de uno o más nombres de variables, finalizando con un punto y coma.

Ejemplo:

```
int a, b, c;
float x, y, z;
short int j;
long int r, s;
```

3.2.2: RECONOCE LOS OPERADORES ARITMÉTICOS, DE ASIGNACIÓN Y RELACIONALES EN LENGUAJE C.

En C, los operadores aritméticos pueden ser de dos tipos, binarios y unarios:

Los binarios:

```
+      suma.
-      resta.
```

* multiplicación.
/ división.
% módulo (resto)

Sintaxis:

Variable_1 operador variable_2

Los unarios:

++ incremento (suma 1)
-- decremento (resta 1)
- cambio de signo.

Sintaxis:

Variable operador
Operador variable

Es necesario tener en cuenta que el resultado de las expresiones depende del tipo de dato de las variables.

Por ejemplo si realiza una división con dos variables enteras, el resultado será también un número entero. Ejemplo $a=13$, $b=2$, entonces $a/b = 6$

En caso de que a y b sean declarados de tipo float, el resultado de a/b sería 6,5.

En C existen varios operadores de asignación, todos se utilizan para formar expresiones de asignación, en las que se asigna el valor de una expresión a un identificador.

El operador más usado es $=$. Las expresiones que usan este operador tienen la siguiente forma:

Identificador = expresión

Donde identificador representa generalmente una variable y expresión una constante, variable o una expresión más compleja.

Ejemplos:

$a = 3$
 $x = y$
 $\text{suma} = a + b$

En C están permitidas las asignaciones múltiples, esto se realiza de la siguiente forma:

Identificador 1 = identificador 2 = = expresión

Ejemplos:

$a = b = c = 2$

Esto sería equivalente a:

a = 2
b = 2
c = 2

También existen otros operadores de asignación de la forma +=, -=, *=, estos se usan de la siguiente forma:

Expresión 1 += expresión 2

Que reemplazaría a:

Expresión 1 = expresión 1 + expresión 2

El resto funciona de la misma forma.

Ejemplos:

Expresión	Expresión Equivalente
i += 5	i = i + 5
j -= 1	j = j - 1
a *= (x + y)	a = a * (x + y)

Los operadores relacionales soportados por C son los siguientes:

Operador	Significado
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
= =	Igual que
!=	Distinto

Estos operadores forman expresiones lógicas, sin embargo el resultado será de tipo entero, ya que verdadero se representa por el valor entero 1, y falso por el valor 0.

Ejemplos: suponga que i, j, k, son variables enteras con valores asignados 1,2,3, respectivamente, entonces:

Expresión	Resultado Lógico	Valor
i < j	Verdadero	1
(i + j) >= k	Verdadero	1
(j + k) > (i + 5)	Falso	0
k != 3	Falso	0
j == 2	verdadero	1

En conjunto con los operadores relacionales C tiene 3 operadores lógicos, estos se denominan “y” lógica, “o” lógica y el operador unario “no” lógico:

Operador	Significado
&&	Y lógica
	O lógica
!	No lógico

Ejemplos: suponga que i, j, k, son variables enteras con valores asignados 1,2,3, respectivamente, entonces:

Expresión	Resultado Lógico	Valor
(i < j) && (k == j)	Falso	0
((i + j) >= k) (k != 3)	Verdadero	1
!(j == 2)	Falso	0

Además de los operadores vistos, C posee una serie de funciones de biblioteca que realizan operaciones y cálculos de uso frecuente. Estas funciones no son parte del lenguaje en si, pero las incluyen todas las implementaciones del lenguaje.

Las funciones de biblioteca se agrupan según propósitos similares, en archivos. Estos archivos se proporcionan como parte de cada compilador de C.

Para poder acceder a estos archivos de funciones se usa la instrucción del preprocesador `#include`, cuya sintaxis es:

```
#include <nombre_archivo>
```

Donde `nombre_archivo`, corresponde al nombre del archivo de funciones de biblioteca.

3.2.3: RECONOCE LA SINTAXIS DE LAS INSTRUCCIONES DE ENTRADA Y SALIDA EN LENGUAJE C.

Sentencia `printf()`

La rutina `printf` permite la aparición de valores numéricos, caracteres y cadenas de texto por pantalla.

El prototipo de la sentencia *printf* es el siguiente:

```
printf(control,arg1,arg2...);
```

La cadena de control indica la forma en que se mostrarán los argumentos posteriores. También puede introducir una cadena de texto (sin necesidad de argumentos), o combinar ambas posibilidades, así como secuencias de escape.

En caso de utilizar argumentos debe indicar en la cadena de control tantos modificadores como argumentos vaya a presentar.

El modificador está compuesto por el caracter % seguido por un caracter de conversión, que indica de que tipo de dato se trata.

Los modificadores más utilizados son:

%c	Un único carácter.
%d	Un entero con signo, en base decimal.
%u	Un entero sin signo, en base decimal.
%o	Un entero en base octal.
%x	Un entero en base hexadecimal.
%e	Un número real en coma flotante, con exponente.
%f	Un número real en coma flotante, sin exponente.
%s	Una cadena de caracteres
%p	Un puntero o dirección de memoria.

Ejemplo:

```
/* Uso de la sentencia printf() 1. */
#include <stdio.h>
main()
/* Muestra por pantalla una suma */
{
    int a=20,b=10;
    printf("El valor de a es %d\n",a);
    printf("El valor de b es %d\n",b);
    printf("Por tanto %d+%d=%d",a,b,a+b);
}
```

Sentencia scanf()

La rutina scanf permite entrar datos en la memoria del computador a través del teclado.

El prototipo de la sentencia scanf es el siguiente:

```
scanf(control,arg1,arg2...);
```

En la cadena de control se indica, por regla general, los modificadores que harán referencia al tipo de dato de los argumentos. Al igual que en la sentencia printf los modificadores estarán formados por el caracter % seguido de un caracter de conversión. Los argumentos indicados serán, nuevamente, las variables.

La principal característica de la sentencia scanf es que necesita saber la posición de la memoria del computador en que se encuentra la variable para poder almacenar la información obtenida. Para indicarle esta posición utilice el símbolo ampersand (&), que se coloca delante del nombre de cada variable. (Esto no será necesario en los arrays).

Ejemplo:

```
/* Uso de la sentencia scanf(). */
#include <stdio.h>
main()
/* Solicita dos datos */
{
    int edad;
    printf("Introduce tu edad: ");
    scanf("%d",&edad);
    printf("tu edad es: %d", edad);
    printf("tu edad el proximo año sera: %d", ++edad);
}
```

Nota: Existen otras instrucciones de entrada y salida de datos, puntualmente para el ingreso o salida de un carácter (getchar() y putchar()), La investigación de ellas puede ser sugerida como trabajo, taller o investigación para los alumnos.

3.2.4: RECONOCE LA SINTAXIS DE LAS INSTRUCCIONES DE DECISION EN LENGUAJE C.

Este tipo de sentencias permiten variar el flujo del programa en base a unas determinadas condiciones. Existen varias estructuras diferentes:

Instrucción IF...ELSE

Sintaxis:

if (condición) sentencia; La sentencia solo se ejecuta si se cumple la condición. En caso contrario el programa sigue su curso sin ejecutar la sentencia

if (condición) sentencia1; else sentencia2; Si se cumple la condición ejecutará la sentencia1, sinó ejecutará la sentencia2. En cualquier caso, el programa continuará a partir de la sentencia2.

Ejemplo 1:

```
/* Uso de la sentencia condicional IF. */
#include <stdio.h>
main()
/* Simula una clave de acceso */
{
    int usuario,clave=18276;
    printf("Introduce tu clave: ");
    scanf("%d",&usuario);
    if(usuario==clave) printf("Acceso permitido");
    else printf("Acceso denegado");
}
```

Ejemplo 2:

```
/* Uso de los op. lógicos AND,OR,NOT. */
#include <stdio.h>
main()
/* Compara un número introducido */
{
    int numero;
    printf("Introduce un número: ");
    scanf("%d",&numero);
    if(!(numero>=0)) printf("El número es negativo");
    else if((numero<=100)&&(numero>=25)) printf("El número está entre 25 y 100");
    else if((numero<25)||((numero>100))) printf("El número no está entre 25 y 100");
}
```

Instrucción switch

Esta instrucción hace que se seleccione un grupo de instrucciones entre varios grupos disponibles. La selección se basa en el valor de una expresión que se incluye en la instrucción switch.

Se suele utilizar en los menús, de manera que según la opción seleccionada se ejecuten una serie de sentencias.

Su sintaxis es:

```
switch (expresión)
{
    case expresión1: instrucciones;
    break;
    case expresión2: instrucciones;
    break;
    default: instrucciones;
}
```

Consideraciones especiales:

- Cada case puede incluir una o más instrucciones sin necesidad de ir entre llaves, ya que se ejecutan todas hasta que se encuentra la sentencia **break**.
- La expresión evaluada sólo puede ser de tipo entero o caracter.
- default ejecutará las sentencias que incluya, en caso de que la opción escogida no exista.

Ejemplo:

```
/* Uso de la sentencia condicional SWITCH. */
#include <stdio.h>
main()
```



```
{
    int dia;
    printf("Introduce el número del día: ");
    scanf("%d",&dia);
    switch(dia)
    {
        case 1: printf("Lunes");
        break;
        case 2: printf("Martes");
        break;
        case 3: printf("Miércoles");
        break;
        case 4: printf("Jueves");
        break;
        case 5: printf("Viernes");
        break;
        case 6: printf("Sábado");
        break;
        case 7: printf("Domingo");
        break;
        default: printf("no existe ese día");
    }
}
```

3.2.5: RECONOCE LA SINTAXIS DE LAS INSTRUCCIONES DE REPETICION EN LENGUAJE C.

En C se pueden utilizar varias instrucciones de ciclo o repetición, estas estructuras que permiten ejecutar partes del código de forma repetida mientras se cumpla una condición. La condición puede ser simple o compuesta de otras condiciones unidas por operadores lógicos.

A continuación se presentan las instrucciones de ciclo más comunes: While, Do-while y For:

Instrucción WHILE:

Su sintaxis es:

```
while (expresión) instrucciones;
```

Con esta sentencia se controla la condición antes de entrar en al ciclo. Si ésta no se cumple, el programa no entrará en ciclo. Naturalmente, si en el interior del ciclo hay más de una sentencia, éstas deberán ir entre llaves para que se ejecuten como un bloque.

Ejemplo 1:

```
/* Uso de la sentencia WHILE. */
#include <stdio.h>
main()
/* Escribe los números del 1 al 10 */
{
    int numero=1;
    while(numero<=10)
    {
        printf("%d\n",numero);
        numero++;
    }
}
```

Ejemplo 2:

```
/*Calcular el promedio de n numeros*/
#include <stdio.h>
main()
{
    int n, cont = 1;
    float x, prom, suma = 0;
    printf("Cuántos números?");
    scanf("%d", &n)
    while (cont <= n) {
        printf("x = ");
        scanf("%f", &x);
        suma += x;
        ++cont;
    }
    Prom = suma/n
    Printf("\n El promedio es %f\n", prom);
}
```

Instrucción DO...WHILE

Su sintaxis es:

Do

```
{
    sentencia1;
    sentencia2;
}
while (condición);
```

Con esta sentencia se controla la condición al final del ciclo. Si ésta se cumple, el programa vuelve a ejecutar las sentencias del ciclo.

La única diferencia entre las sentencias while y do...while es que con la segunda el cuerpo del ciclo se ejecutará por lo menos una vez.

Ejemplo: El siguiente programa muestra un menú de opciones en pantalla, mientras no se digite el nº 4:

```
/* Uso de la sentencia DO...WHILE. */
#include <stdio.h>
main()
/* Muestra un menú si no se pulsa 4 */
{
char seleccion;
do
{
printf("1.- Comenzar\n");
printf("2.- Abrir\n");
printf("3.- Grabar\n");
printf("4.- Salir\n");
printf("Escoge una opción: ");
seleccion=getchar();
switch(seleccion)
{
case '1': printf("Opción 1");
break;
case '2': printf("Opción 2");
break;
case '3': printf("Opción 3");
break;
}
}
while(seleccion!='4');
}
```

Instrucción FOR

Su sintaxis es:

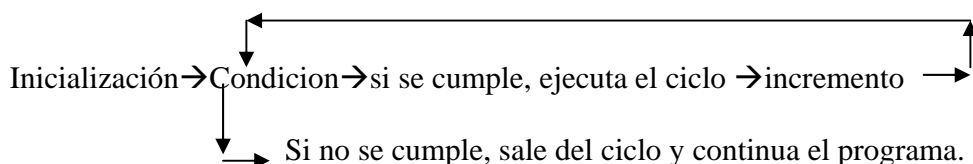
```
for (inicialización;condición;incremento)
{
sentencia1;
sentencia2;
}
```

La inicialización indica una variable (variable de control) que condiciona la repetición del ciclo. Si hay más, van separadas por comas:

Ejemplo:

```
for (a=1,b=100;a!=b;a++,b- -)
{
```

El flujo del ciclo FOR transcurre de la siguiente forma:



Ejemplo:

```
/* Uso de la sentencia FOR. */
#include <stdio.h>
main()
/* Escribe la tabla de multiplicar */
{
    int num,x,result;
    printf("Introduce un número: ");
    scanf("%d",&num);
    for (x=0;x<=10;x++)
    {
        result=num*x;
        printf("\n%d por %d = %d\n",num,x,result);
    }
}
```

Instrucción CONTINUE

Se utiliza dentro de un ciclo. Cuando el programa llega a una sentencia CONTINUE no ejecuta las líneas de código que hay a continuación y salta a la siguiente iteración del ciclo.

Ejemplo:

```
/* Uso de la sentencia CONTINUE. */
#include <stdio.h>
int main()
/* Escribe los números del 1 al 100 menos el 25 */
{
    int numero=1;
    while(numero<=100)
    {
        if (numero==25)
        {
            numero++;
            continue;
        }
    }
}
```

```

    }
    printf("%d\n",numero);
    numero++;
}
}

```

3.2.6. CODIFICA PROGRAMAS FUENTE EN LENGUAJE C, SEGÚN SU CORRESPONDIENTE PSEUDOLENGUAJE.

Convertir los algoritmos en pseudolenguaje a C.

1.- Algoritmo que permite calcular el promedio aritmético de 7 notas.

```

START
NUM NOTA, C, S, P
S:=0
C:=0
DO
READ "INGRESE NOTA", NOTA
S:=S+NOTA
C:=C+1
DO_WHILE (C<7)
P:=S/7
PRINT "EL PROMEDIO ES :",P
END

```

Solución:

```

#include <stdio.h>

main()
{
    int C;
    float NOTA,P,S;
    S=0;
    C=0;
    do{
        printf("INGRESE NOTA ");
        scanf("%f",&NOTA);
        S=S+NOTA;
        C++;
    }
    while(C<7);
    P=S/7;
    printf("EL PROMEDIO ES :%f\n",P);
    system("PAUSE");
}

```

2.- Algoritmo que permite ingresar 15 números enteros negativos y los convierte en positivos:

```
START
NUM N, C, P
C:=0
DO
DO
READ "INGRESE UN NUMERO NEGATIVO:", N
DO_WHILE (N>=0)
P:=N*-1
PRINT P
C:=C+1
DO_WHILE (C<15)
END
```

Solución:

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

```
main()
{
int N, C, P;
C=0;
do{
do{
printf("INGRESE UN NUMERO NEGATIVO: ");
scanf("%d",&N);
}
while (N>=0);
P=N*-1;
printf(" EL NÚMERO EN POSITIVO ES: %d\n",P);
C++;
}
while (C<15);
system("pause");
}
```

3.- Algoritmo que permite ingresar 20 numeros enteros y entrega la cantidad de números que son positivos, negativos y cero.

```

START
NUM I, POS, NEG, NEU, N
I = 1
POS = 0
NEG = 0
NEU = 0
WHILE ( I <= 20)
    PRINT “INGRESE UN NÚMERO”
    READ N
    IF (N = 0) THEN
        NEU = NEU + 1
    ELSE
        IF (N > 0) THEN
            POS = POS + 1
        ELSE
            NEG = NEG + 1
        ENF_IF
    END_IF
    I = I + 1
END_WHILE
PRINT “CANTIDAD DE NÚMEROS POSITIVOS: “, POS
PRINT “CANTIDAD DE NÚMEROS NEGATIVOS: “, NEG
PRINT “CANTIDAD DE NÚMEROS NEUTROS: “, NEU
END

```

Solución:

```

#include <stdio.h>
main()
{
    int i=1, pos=0, neg=0, neu=0, n;
    while (i<=20)
    {
        printf("ingrese un número : ");
        scanf("%d",&n);
        if (n==0) neu++;
        else
        {
            if (n>0) pos++;
            else neg++;
        }
        i++;
    }
    printf("cantidad de números positivos : %d\n", pos);
    printf("cantidad de números negativos : %d\n", neg);
    printf("cantidad de números cero : %d\n", neu);
    system("pause");
}

```

4.- Algoritmo que permite conocer el día del mes de mayo, donde se registró la mayor cantidad de lluvia caída. Asumiendo que se ingresan cantidades diferentes cada día.

```

START
NUM LLUVIA, DIA, ALTA, D
FOR (DIA := 1 ; DIA <=31 ; DIA := DIA + 1)
    READ “INGRESE CANTIDAD DE LLUVIA :”, LLUVIA
    IF (DIA = 1) THEN
        ALTA := LLUVIA
        D := DIA
    ELSE
        IF ( LLUVIA > ALTA) THEN
            ALTA := LLUVIA
            D := DIA
        END_IF
    END_IF
END_FOR
PRINT “EL DIA MAS LLUVIOSO FUE :”, D, “DE MAYO”
END

```

Solución:

```

#include <stdio.h>
main()
{
    int lluvia, dia, alta, d;
    for(dia=1;dia<=31;dia++)
    {
        printf("ingrese la cantidad de agua caída el dia %d de mayo :",dia);
        scanf("%d",&lluvia);
        if (dia==1)
        {
            alta=lluvia;
            d=dia;
        }
        else
        {
            if (lluvia>alta)
            {
                alta=lluvia;
                d=dia;
            }
        }
    }
    printf("el %d de mayo fue el dia que cayo mayor lluvia\n", d);
    system("pause");
}

```


Notas:

Las soluciones presentadas fueron desarrolladas en Dev C++.

Como ejercicio puede pedir a los alumnos que codifiquen en C, los algoritmos realizados en la unidad 2. (ejercicios propuestos de ciclos (objetivo 2.6.6))

EJERCICIOS PROPUESTOS DE LENGUAJE C (OBJETIVO 3.2.6)

1.- Realizar un programa en C, que permita convertir una temperatura leída en grados Fahrenheit a grados Celsius, para ello debe usar la fórmula:

$$C = 0,556 * (F - 32)$$

2.- Construir un programa en C, para calcular el volumen y el área de una esfera, las fórmulas para ello son:

$$V = 4\pi r^3/3$$

$$A = 4\pi r^2$$

3.- Realice un programa en C, que permita calcular x^y , usando multiplicaciones sucesivas.

4.- Realice un programa que permita mostrar por pantalla un menú de opciones con las siguientes acciones:

1. Calcular la sumatoria de 10 números enteros ingresados.
2. Calcular la edad de una persona ingresando el año de nacimiento.
3. salir del programa.

Y que realice cada una de ellas, según lo que el usuario elija, el programa debe terminar solo si el usuario elige la opción 3.

5.- Modifique el programa nº1, para que a partir de una temperatura en grados Celsius, se convierta en grados Fahrenheit. ($F = C * 1,8 + 32$)

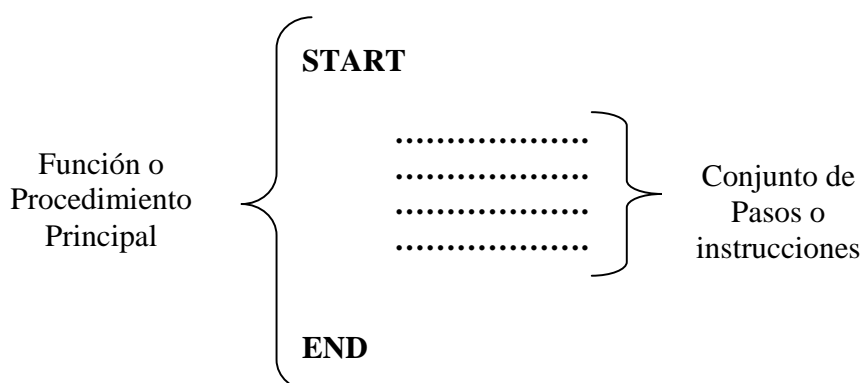
UNIDAD 4: SUB-PROCESOS

OBJETIVO 4.1: DESCRIBE EL PROCESO DE PROGRAMACIÓN MODULAR

4.1.1: IDENTIFICA LA ESTRUCTURA GENERAL DE UN PROGRAMA

Hasta Ahora los algoritmos que se han escrito tienen una única estructura que pasa desapercibida ante la vista del estudiante, dado el interés por resolver los problemas, impide ver que éstos se han estado desarrollando dentro de un “**Módulo**” principal. Pues bien, ésta unidad denominada “**SUB-PROCESOS**” permitirá al estudiante comprender más acerca de la estructura de sus algoritmos como partes funcionales o “Módulos” que pueden ser invocados desde un “Módulo” principal con el objetivo de cumplir tareas específicas y de simplificar la complejidad de algunos algoritmos por medio de una conocida estrategia utilizada comúnmente en informática denominada “*Divide y Vencerás*”.

Un programa se desarrolla dentro de una función o procedimiento principal la cual es una estructura que es invocada automáticamente o “por defecto” al momento de iniciar el procesamiento de las órdenes o instrucciones que han sido escritas en su interior por el programador.



Dicha estructura puede contener “sub-conjuntos” de pasos o instrucciones que tal vez serán necesarias o “requeridas” muchas veces exactamente como han sido escritas (o con cambios menores) pero en distintos lugares “lógicos” de nuestro algoritmo. A primera vista lo que se acaba de mencionar puede parecer difícil de entender por lo que el siguiente ejemplo ayudará a despejar dudas:

Problema:

“Se necesita un algoritmo que calcule el promedio de edades de los niños, niñas, hombres y mujeres de un universo de 50 personas.”

Solución en pseudolenguaje:

Start

Num x,per,edad,mujer,hombre,nina,nino

Num Nmujer,Nhombre,Nnina,Nnino

Num Pmujer,Phombre,Pnina,Pnino

Nhombre := Phombre := Nmujer := Pmujer Nnina := Pnina := Nnino := Pnino := 0

Print "Ingreso de personas para calculo de promedio de edades"

Print "=====

Print "Para ingresar digite :"

Print " Hombre = 1, Mujer = 2, Niña = 3, Niño = 4 "

For (x := 1 , x < 50 , x := x + 1)

Do

Read "ingrese persona",per

Do_While(per < 1 or per > 4)

Read "ingrese edad", edad

Select per in

**Case 1 : Nhombre := Nhombre +1
hombre := hombre + edad
break**

**Case 2 : Nmujer := Nmujer +1
mujer := mujer + edad
break**

**Case 3 : Nnina := Nnina +1
nina := nina + edad
break**

**Case 4 : Nnino := Nnino +1
nino := nino + edad
break**

End_Case

End_For

Phombre := hombre / Nhombre

Pmujer := mujer / Nmujer

Pnina := nina / Nnina

Pnino := nino / Nnino

Print "El promedio de edad de Hombres es ",Phombre

Print "El promedio de edad de Mujeres es ",Pmujer

Print "El promedio de edad de Niñas es ",Pnina

Print "El promedio de edad de Niños es ",Pnino

End

Solución en C:

```
#include <stdio.h>
int main()
{
    int per,x,edad,mujer,hombre,nina,nino,Nmujer,Nhombre,Nnina,Nnino;
    float Pmujer,Phombre,Pnina,Pnino;
    Nhombre=Phombre=Nmujer=Pmujer=Nnina=Pnina=Nnino=Pnino=0;
    edad=mujer=hombre=nina=nino=0;
    printf("Ingreso de personas para calculo de promedio de edades\n");
    printf("=====\\n");
    for(x=1; x <50;x++)
    {
        do
        {
            printf("Para ingresar digite :\\n");
            printf(" Hombre = 1, Mujer = 2, Niña = 3, Niño = 4 \\n");
            scanf("%d",&per);
        }
        while(per<1 || per>4);
        printf("Ingrese la edad :\\n");
        scanf("%d",&edad);
        switch(per){
            case 1:
                Nhombre++;
                hombre= hombre + edad;
                break;
            case 2:
                Nmujer++;
                mujer= mujer + edad;
                break;
            case 3:
                Nnina++;
                nina= nina + edad;
                break;
            case 4:
                Nnino++;
                nino= nino + edad;
                break;
        }
    }
    Phombre= hombre / Nhombre;
    Pmujer= mujer / Nmujer;
    Pnina= nina / Nnina;
    Pnino= nino / Nnino;
    printf("El promedio de edad de Hombres es %.0f\\n",Phombre);
    printf("El promedio de edad de Mujeres es %.0f\\n",Pmujer);
    printf("El promedio de edad de Niñas es %.0f\\n",Pnina);
    printf("El promedio de edad de Niños es %.0f\\n",Pnino);
}
```

Nota: Este programa fue creado en Dev C++, además funciona siempre que exista al menos una persona en cada categoría, de lo contrario se realiza una división 0/0.

Observando el ejemplo anterior, es posible distinguir rápidamente que existen instrucciones que son iguales en términos de “acciones” a seguir con determinados “argumentos” que varían de acuerdo al contexto (Nótese las cuatro primeras “Case”). La pregunta que cabe darse es:

¿Es posible centralizar en un solo módulo dichas líneas para generar un código más compacto y eficiente en términos de escritura de instrucciones?

¿Es posible invocar a esas instrucciones con distintos “argumentos” para cumplir con los requerimientos de distintos contextos?

¿Es posible, finalmente, que un módulo distinto del principal pueda ser llamado tantas veces como sea requerido con el objeto de escribir solo una vez las instrucciones y de haber modificaciones hacerlas solo en ese determinado módulo?

Para todas las interrogantes la respuesta es **SI**, por medio de un diseño **MODULAR**.

4.1.2: IDENTIFICA LAS PARTES PRINCIPALES DE UN PROCESO MODULAR.

En la etapa de análisis se determina que hace el programa, en la etapa de diseño se determina cómo hace el programa la tarea solicitada. Con el objetivo de hacer el proceso de la resolución de problemas más eficaz es que se utiliza el proceso de diseño conocido como **“divide y vencerás”**. Esto quiere decir que la resolución de problemas complejos se realizan dividiendo dicho problema en subproblemas y a continuación dividir dichos subproblemas en otros de nivel más bajo hasta que pueda ser implementada la solución. La solución de estos subproblemas se realiza con subalgoritmos. Los subalgoritmos son unidades de programa o módulos que están diseñados para ejecutar alguna tarea específica. Estas unidades (las que distinguiremos como “Funciones” y “Procedimientos” y que estudiaremos en profundidad más adelante) se escriben sólo una vez, pero pueden ser invocadas o referenciadas en diferentes puntos del programa principal o “módulo principal” con el objeto de no duplicar el código innecesariamente. Este método de diseñar algoritmos con el proceso de “romper” el problema subdividiéndolo en varios subproblemas se denomina **“diseño descendente”, “TOP-DOWN” o “Modular”** y en cada etapa, expresar cada paso en forma más detallada se denomina **“refinamiento sucesivo”**. Cada subprograma es resuelto en un módulo que tiene un único punto de entrada y un solo punto de salida. Los módulos pueden ser planeados, codificados, comprobados y depurados de forma individual (hasta por programadores distintos) y combinarlos mas tarde lo que implica la combinación de los siguientes pasos:

- 1.- Programar un Módulo
- 2.- Comprobar el Módulo
- 3.- Depurar el Módulo (si aplica)
- 4.- Combinar el Módulo con los demás.

La programación modular es uno de los métodos de diseño más flexible y potentes para mejorar la productividad de un programa.

4.1.3: RECONOCE LA RELACIÓN ENTRE PROGRAMACIÓN ESTRUCTURADA Y PROGRAMACIÓN MODULAR.

Los términos *Programación Modular*, *Programación descendente* y *Programación Estructurada* se introdujeron en la segunda mitad de la década de los sesenta y a menudo sus términos se utilizan como sinónimos aunque no significan lo mismo.

La programación modular y descendente (como se ha mencionado) consiste en la descomposición de un programa en módulos independientes más simples (lo que se conoce como el método “*divide y vencerás*”), donde se diseña cada módulo con independencia de los demás, y siguiendo un método descendente se llegará hasta la descomposición final del problema en módulos de forma jerárquica.

La programación estructurada significa escribir un programa de acuerdo a las siguientes reglas. Se refiere a un conjunto de técnicas que han ido evolucionando desde los primeros trabajos de Edgar Dijkstra¹. Estas técnicas aumentan considerablemente la productividad del programa reduciendo bastante el tiempo requerido para escribir, verificar, depurar y mantener los programas. Utiliza un número limitado de estructuras de control que minimizan la complejidad de los programas y reducen la posibilidad de error.

Las reglas de la programación estructurada son las siguientes:

- El programa tiene un diseño Modular
- Los módulos son diseñados de modo descendente
- Cada módulo se codifica utilizando las tres estructuras de control básicas :
 1. Secuencia
 2. Selección
 3. Repetición

La programación estructurada es el conjunto de técnicas que incorporan lo siguiente:

- Recursos Abstractos

Descomponer una determinada acción compleja en términos de número de acciones más simples capaces de ejecutarlas o que constituyan instrucciones de computadoras disponibles
- Diseño descendente (Top-down) :

Es el proceso mediante el cual un problema se descompone en una serie de niveles o pasos sucesivos de refinamiento. Se descompone el problema en etapas o estructuras jerárquicas , de forma que se puede considerar cada estructura desde los puntos de vista :¿Qué hace? Y ¿Cómo lo hace?.
- Estructuras Básicas

OBJETIVO 4.2: DESCOMPONE UN PROBLEMA GENERAL EN SUB-PROBLEMAS O MÓDULOS.

4.2.1: RECONOCE CUANDO ES NECESARIO MODULARIZAR UN PROBLEMA GENERAL.

La resolución de problemas complejos se facilita considerablemente si se dividen en problemas más pequeños (subproblemas). La solución de estos subproblemas se realiza con subalgoritmos. El uso de subalgoritmos permite al programador desarrollar programas de problemas complejos utilizando el método descendente introducido en los ítems anteriores.

Consideremos dos problemas que permitirán por sus características comprender el reconocimiento de un problema que se debe modularizar:

1. En el ítem 4.1.1 se destaca que los cuatro primeros conjuntos de casos a analizar tienen instrucciones muy similares que pudieran ser reducidas por medio de un único módulo el cual sólo cambiaría los datos de entrada a saber :

```

Case 1 :   Nhombre := Nhombre +1
           hombre := hombre + edad
           break
Case 2 :   Nmujer := Nmujer +1
           mujer := mujer + edad
           break
Case 3 :   Nnina := Nnina +1
           nina := nina + edad
           break
Case 4 :   Nnino := Nnino +1
           nino := nino + edad
           break
    
```

Si reflexionamos respecto de la solución dada (paso 4 de las estrategias de resolución de problemas según Polya “Mirar hacia atrás”), es posible observar que cada instrucción respecto de las alternativas de selección del “*Select*”, depende del único dato ingresado en cada iteración (Tipo de persona y edad) para luego efectuar el cálculo del incremento de la cantidad de personas de su mismo tipo y de la suma de las edades. ¿Es posible una forma más eficiente?. La respuesta es si, por medio de una invocación a un único módulo al cual darle la referencia de las variables respectivas en todas las alternativas posibles de la estructura de selección múltiple.

2. Consideremos el cálculo de la superficie (área) de un rectángulo. ¿Este problema se puede dividir? La respuesta es si, en dos subproblemas :
 - Subproblema 1 : Entrada de valores para base y altura, salida de resultados
 - Subproblema 2 : Cálculo de superficie

Asúmase que estos ejemplos pretenden dar una “aclaratoria” respecto de la modularización con el objetivo de entender cabalmente los conceptos y tómese tales ejemplos como analogía de problemas más complejos como los siguientes:

1. “Para cada departamento de cada piso de cada edificio existente en un cuadrante determinado, ingresar el Nombre y Rut de su respectivo propietario y listar los nombres y edades de los integrantes del respectivo grupo familiar. Además establezca el promedio de edad de los integrantes menores de 18 años.”
2. “Programa que para el ingreso de 8 tripletas de números retorne cada una de dichas tripletas en un orden ascendente”

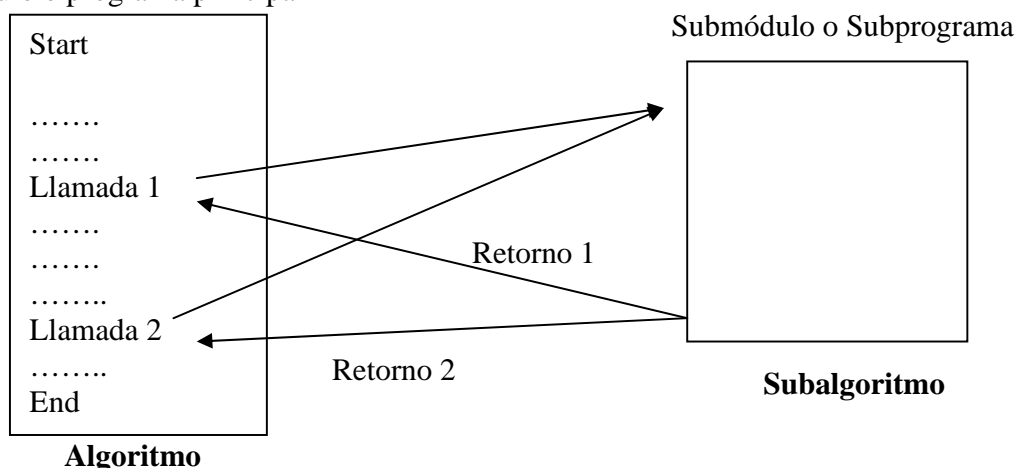
4.2.2: DESCOMPONE UN PROBLEMA GENERAL EN SUBPROBLEMAS O MÓDULOS.

Un método ya citado para solucionar un problema complejo es dividirlo en subproblemas más sencillos y a continuación dividir estos subproblemas en otros más simples hasta que los problemas más pequeños sean fáciles de resolver. El problema principal se soluciona por el correspondiente **Programa** o **Algoritmo Principal** (también denominado controlador o conductor) y la solución de los subproblemas mediante **Subprogramas**, conocidos como **Procedimientos (Subrutinas)** o **funciones**.

Un subprograma puede realizar las mismas acciones que un programa: Aceptar datos, Realizar cálculos, Retornar resultados; sin embargo, los subprogramas son utilizados para un propósito específico: “recibir datos desde el programa y retornar resultados cuando se requiere”.

Cada programa contiene un módulo denominado “*programa principal*”, que controla todo lo que sucede; se transfiere el control a *submódulos o subprogramas*, ellos ejecutan sus instrucciones y posteriormente estos devuelven el control al programa principal.

Módulo o programa principal



Nótese que es posible que existan varios submódulos y eso depende de las necesidades que impone el problema.

4.2.3: RECONOCE LOS TIPOS DE SUBMÓDULOS Y SUS ESTRUCTURAS

Existen dos tipos de submódulos bien definidos que se detallan a continuación:

FUNCIONES: Una función es una operación que toma uno o más valores llamados “Argumentos”, los cuales procesa por medio de sus instrucciones internas y retorna un sólo valor.

La declaración de una función requiere una serie de pasos que la definen. Como Subalgoritmo o Subprograma tiene una constitución similar a los algoritmos, por lo que constará de una cabecera que comenzará por el tipo de valor devuelto por la función (**Num** para valor numérico y **Char(n)** para valores alfanuméricos en donde “n” indica el número de caracteres), seguido del nombre de la función y los argumentos de dicha función. A continuación irá el cuerpo de la función, que es la serie de acciones entre las que se destaca la acción **return(expresión)** que es la acción de retornar un valor por el cual fue invocada dicha función y en donde **expresión** es la expresión aritmética o el valor constante (número) o el valor de una variable. Finalmente la función terminará con **End_nombre_función**.

Su sintaxis es la siguiente:

(tipo_de_valor_de_retorno) Nombre_de_función (lista de parámetros)

declaración de variables locales
instrucciones
return(expresión) ←

Num/ Char(n)
(De éste parámetro
depende (expresión))

End_Nombre_de_función

En donde la **lista de parámetros** está compuesta de:

(tipo_de_transferencia tipo_de_variable nombre_variable....)

Byval/Byref

Num/ Char(n)

Tipo_de_valor_de_retorno	:	Es el tipo de dato que retornará la función
Tipo_de_transferencia	:	Es la forma que se pasarán los parámetros y puede tomar solo dos valores Byval y Byref los cuales se estudiarán mas adelante.

Ejemplo:

Este ejemplo consiste en una función que es llamada desde el programa principal con dos argumentos: La base y la altura. Dicha función retornará el cálculo del área a la función principal. (Ver problema 2 en el ítem 4.2.1)

```

Num Calcula_área ( byval num base, byval num altura)
    Num area
    Area := base * altura
    Return(area)
End_calcula_área
    
```

Para el lenguaje C, las funciones son bloques de código utilizados para dividir un programa en partes más pequeñas, cada una de las cuáles tendrá una tarea determinada.

Su sintaxis es:

```
tipo_función nombre_función (tipo y nombre de argumentos)  
{  
    bloque de sentencias  
}
```

Donde:

tipo_función: puede ser de cualquier tipo de los que conocemos. El valor devuelto por la función será de este tipo. Por defecto, es decir, si no indicamos el tipo, la función devolverá un valor de tipo entero (**int**) (Debe revisar si en la versión de C usada es soportado esto o no). Si no queremos que retorne ningún valor deberemos indicar el tipo vacío (**void**).

nombre_función: es el nombre que le daremos a la función.

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.

bloque de sentencias: es el conjunto de sentencias que serán ejecutadas cuando se realice la llamada a la función.

Nota: Las funciones pueden ser llamadas desde la función **main** o desde otras funciones. Nunca se debe llamar a la función **main** desde otro lugar del programa. También es necesario declarar el prototipo de la o las funciones, es decir antes del programa principal debe ir el encabezado de la función. Por último recalcar que los argumentos de la función y sus variables locales se destruirán al finalizar la ejecución de la misma.

Ejemplo:

Este ejemplo consiste en una función que es llamada desde el programa principal con dos argumentos: La base y la altura. Dicha función retornará el cálculo del área a la función principal. (Ver problema 2 en el ítem 4.2.1)

```
int calcula_area(int a, int b)  
{  
    int r;  
    r= a*b;  
    return(r);  
}
```

Nota: Para este ejemplo se consideró que los valores de base, altura y area son Enteros.

PROCEDIMIENTOS: Un procedimiento es un subprograma que ejecuta un proceso específico y ningún valor está asociado con el nombre de dicho procedimiento, por consiguiente no existe un valor de retorno.

Su sintaxis es la siguiente:

```
Nombre_procedimiento (lista de parámetros)
    declaración de variables locales
    instrucciones
End_Nombre_procedimiento
```

Ejemplo:

Este ejemplo consiste en un procedimiento que es llamado desde el módulo o programa principal y realiza las tareas de actualizar valores en variables para el cálculo de promedio de edades en base a cierto criterio. (Ver problema 1 de ítem 4.2.1)

```
Promedia (byref num numero, byref num tipo, byval num edad)
    Numero := numero + 1
    Tipo := tipo + edad
End_promedia
```

En Lenguaje C, no existe el procedimiento como tal, sino que las funciones pueden trabajar como si fueran un procedimiento, simplemente se debe indicar que no retornen ningún valor, usando el tipo vacío (void).

Ejemplo:

```
void promedia(int *numero,int *tipo,int edad)
{
    *numero = *numero+1;
    *tipo = *tipo+edad;
}
```

OBJETIVO 4.3: RESUELVE PROBLEMAS HACIENDO USO DE SUBPROGRAMAS, A TRAVES DE PROCEDIMIENTOS Y FUNCIONES

4.3.1 ESTABLECE LA RELACIÓN ENTRE EL MÓDULO PRINCIPAL Y LOS SUBMÓDULOS.

Una vez que se ha internalizado el concepto de modularización y se ha formalizado las sintaxis de las funciones y procedimientos resta establecer la relación que existe entre éstos y el programa principal. El evento de “relacionar” ambas partes (programa principal –función o Programa principal- procedimiento) es en extremo sencillo.

Para llamar desde el programa principal a una **función** debe efectuarse siguiendo las siguientes reglas:

- Relacionar a una variable la invocación de la función (por el tipo de valor de retorno). Es decir, la función debe ir al lado derecho de una asignación.

- Relacionar las variables de argumento del programa con las variables de argumento (lista de parámetros) en la función en su mismo orden.

Así en el ejemplo citado en el ítem anterior (3.2.3) que establece una función que calcula el área y que tiene relación con el problema citado en el ítem 3.2.1, es posible establecer una solución final como sigue:

Problema :

“Consideremos el cálculo de la superficie (área) de un rectángulo. ¿Este problema se puede dividir? La respuesta es sí, en dos subproblemas :

- Subproblema 1 : Entrada de valores para base y altura, salida de resultados
- Subproblema 2 : Cálculo del área”

Solución :

Start

Num bas, alt, resultado

Read “ingrese valor de base”,bas

Read”ingrese valor de altura”,alt

resultado := Calcula_área (bas, alt)

Print “el resultado es “,resultado

End

Calcula_área (byval num base, byval num altura)

Num area

Area := base * altura

Return(area)

End_calcula_área

En C, la solución sería:

```
#include <stdio.h>
/*Prototipo de la función*/
int calcula_area(int,int);
int main()
{
    int base, alt, area;
    printf("ingrese la base : ");
    scanf("%d",&base);
    printf("ingrese la altura : ");
    scanf("%d",&alt);
    area=calcula_area(base,alt);
    printf("El area es : %d\n",area);
    system("PAUSE");
}

int calcula_area(int a, int b)
{
    int r;
    r= a*b;
    return(r);
}
```

De ésta forma debe entenderse que al iniciar el programa se efectúan los siguientes pasos :

- definición de variables (bas, alt, resultado) que son locales al programa principal (ya formalizaremos los conceptos de alcance de variables)
- se efectúan las instrucciones de lectura de valores de las variables,
- luego para asignar a la variable “resultado” se invoca a la función “**Calcula_área**” con los parámetros “bas” y “alt” como argumentos de “Entrada” a la función.
- La función define una variable local (area) la cual recibe el valor del producto entre las variables de argumento de entrada (base y altura que a su vez ha recibido por invocación el valor de las variables locales al programa principal bas y alt)
- retornar el valor calculado en la función y traspasar el control al programa principal.
- Se despliega el resultado
- Finaliza el programa

Para llamar desde el programa principal a un **Procedimiento** debe efectuarse siguiendo las siguientes reglas:

- Invocar al procedimiento por su nombre
- Relacionar las variables de argumento del programa con las variables de argumento (lista de parámetros) en la función en su mismo orden.

Así en el ejemplo citado en el ítem anterior (3.2.3) que establece un procedimiento que calcula el área y que tiene relación con el problema citado en el ítem 3.1.1, es posible establecer una solución final como sigue:

Problema:

“Se necesita un algoritmo que calcule el promedio de edades de los niños, niñas hombres y mujeres de un universo de 50 personas.”

Solución:

Start

Num x,per,edad,mujer,hombre,nina,nino

Num Nmujer,Nhombre,Nnina,Nnino

Num Pmujer,Phombre,Pnina,Pnino

Nhombre := Phombre := Nmujer := Pmujer Nnina := Pnina := Nnino := Pnino := 0

Print “Ingreso de personas para calculo de promedio de edades”

Print”=====”

Print “Para ingresar digite :”

Print “ Hombre = 1, Mujer = 2, Niña = 3, Niño = 4 “

For (x := 1 , x <50 , x := x + 1)

Do

Read “ingrese persona”,per

Do_While(per < 1 or per > 4)

Read “ingrese edad”, edad

Select per in

Case 1 :Promedia (Nhombre, hombre, edad) break

Case 2 : Promedia (Nmujer, mujer, edad) break

Case 3 : Promedia (Nnina, nina, edad) break

Case 4 : Promedia (Nino, nino, edad) break

End_Case

End_For

Phombre := hombre / Nhombre

Pmujer := mujer / Nmujer

Pnina := nina / Nnina

Pnino := nino / Nnino

Print “El promedio de Hombres es “,Phombre

Print “El promedio de Mujeres es “,Pmujer

Print “El promedio de Niñas es “,Pnina

Print “El promedio de Niños es “,Pnino

End

Promedia (byref num numero, byref num tipo, byval num edad)

numero := numero + 1

Tipo := tipo + edad

End_promedia

En C, la solución sería:

#include <stdio.h>

void promedia(int *,int *,int);

main()

{

int per,x,Nmujer,Nhombre,Nnina,Nnino,edad,mujer,hombre,nina,nino;

float Pmujer,Phombre,Pnina,Pnino;

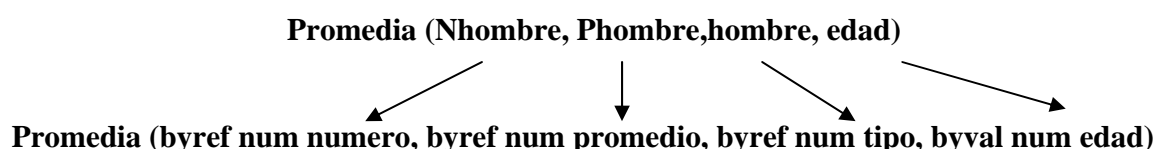
```

Nhombre=Phombre=Nmujer=Pmujer=Nnina=Pnina=Nnino=Pnino=0;
edad=mujer=hombre=nina=nino=0;
printf("Ingreso de personas para calculo de promedio de edades\n");
printf("=====\\n");
for(x=1; x <5;x++)
{
do
{
printf("Para ingresar digite :\\n");
printf(" Hombre = 1, Mujer = 2, Niña = 3, Niño = 4 \\n");
scanf("%d",&per);
}
while(per<1 || per>4);
printf("Ingresa la edad :\\n");
scanf("%d",&edad);
switch(per){
case 1:
promedia(&Nhombre,&hombre,edad);
break;
case 2:
promedia(&Nmujer,&mujer,edad);
break;
case 3:
promedia(&Nnina,&nina,edad);
break;
case 4:
promedia(&Nnino,&nino,edad);
break;
}
}
Phombre= hombre / Nhombre;
Pmujer= mujer / Nmujer;
Pnina= nina / Nnina;
Pnino= nino / Nnino;
printf("El promedio de edad de Hombres es %.0f\\n",Phombre);
printf("El promedio de edad de Mujeres es %.0f\\n",Pmujer);
printf( "El promedio de edad de Niñas es %.0f\\n",Pnina);
printf("El promedio de edad de Niños es %.0f\\n",Pnino);
system("PAUSE");
return 0;
}

void promedia(int *numero,int *tipo,int edad)
{
*numero = *numero+1;
*tipo = *tipo+edad;
}

```

PASO DE PARÁMETROS: En los ejemplos anteriores (tanto en Funciones como en Procedimientos) se ha establecido el paso de parámetros que son variables que pueden ser utilizadas por dichos subprogramas. Pues bien, existen los tipos de transferencia que implican la forma de establecer la transferencia de parámetros. En el ejemplo anterior (de ser cierta la primera selección) se puede establecer que el paso de parámetros es de orden directo y su efecto es como se ve a continuación:



POR VALOR : El paso de parámetros por valor implica que para el subprograma se crea una copia de la variable que se ha traspasado sin afectar el valor de la variable “original”. Únicamente nos interesa el valor, no las modificaciones que pueda tener dentro del subalgoritmo. Son parámetros unidireccionales, que pasan información desde el algoritmo al subalgoritmo. **Puede ser cualquier expresión evaluable en ese momento.** Al estudiar el ejemplo anterior se establece que el procedimiento contiene una transferencia por valor (Byval) de la variable “edad”. De manipular dicha variable en el procedimiento sólo cambiaría la “copia” de la variable en el procedimiento y no en el programa principal. Para mayor claridad se muestra a continuación lo que se denomina el “Heap” (montón de memoria) que grafica el espacio de memoria destinado para cada módulo (contextualizado al ejemplo anterior):

programa Principal				
x	Nhombre	Nmujer	Nnino	Nnina
Phombre	Pmujer	Pnino	Pnina	edad (valor)
mujer	hombre	nino	nina	per
Procedimiento				
numero (Ref)	tipo (Ref)	promedio (Ref)	edad (valor)	

Ejemplo en C, del paso de parámetro por valor:

Este ejemplo realiza un intercambio de valores entre las variables a y b.

```
/* Paso por valor. */
```

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

```
void intercambio(int,int);
```

```
main() /* Intercambio de valores */
```

```
{
```

```
    int a=1,b=2;
```

```
    printf("valores iniciales son a=%d y b=%d\n",a,b);
```



```

intercambio(a,b); /* llamada */
printf("Despues de la funcion intercambio  a=%d y b=%d\n",a,b);
system("pause");
}

void intercambio (int x,int y)
{
    int aux;
    aux=x;
    x=y;
    y=aux;
    printf("En la funcion los valores son  a=%d y b=%d\n",x,y);
}

```

En este ejemplo se puede ver que al pasar las variables por valor, lo que realiza la función con ellas solo queda en la función, por lo tanto su contenido en el programa principal no se ve alterado.

POR REFERENCIA: El paso de parámetros por referencia implica que para el subprograma se crea una copia de la variable que se ha traspasado con la dirección afectando el valor de la variable “original”, vale decir, se pasa una referencia a la posición de memoria donde se encuentra dicho valor. Se utilizan tanto para recibir como para transmitir información sobre el algoritmo y el subalgoritmo. **Debe ser obligatoriamente una variable.**

Al estudiar el ejemplo anterior se establece que el procedimiento contiene una transferencia por referencia a las variables “numero”, “tipo”, “promedio” que al momento de ser llamada la función o procedimiento, lo que reciben las variables del argumento es la dirección de las variables originales en su respectivo orden. De manipular dichas variables en el procedimiento cambiarían las variables originales. Para mayor claridad se muestra a continuación lo que sucedería de ser cierta la segunda proposición de la estructura de control de selección “**Case 2 : Promedia (Nmujer, Pmujer,mujer, edad)**”

programa Principal				
x	Nhombre	Nmujer	Nnino	Nnina
Phombre	Pmujer	Pnino	Pnina	edad (valor)
mujer	hombre	nino	nina	per
Procedimiento				
numero (Ref)	tipo (Ref)	promedio (Ref)	edad (valor)	

El llamado al procedimiento “**Promedia (Nmujer, Pmujer,mujer, edad)**”, traspasa la dirección de sus argumentos (**Nmujer, Pmujer,mujer**) a las variables “numero, promedio y tipo” respectivamente. Nótese que la variable “edad” se traspasa por valor, creándose en el espacio de memoria del procedimiento una copia idéntica de dicha variable que reside en el espacio de memoria del programa principal.

Ejemplo en C, del paso de parámetro por Referencia:

Este ejemplo realiza un intercambio de valores entre las variables a y b, pero ahora se usa el paso de parámetros por referencia.

```
/* Paso por referencia. */

#include <stdio.h>

void intercambio(int *,int *);
main() /* Intercambio de valores */
{
    int a=1,b=2;
    printf("valores iniciales  a=%d y b=%d\n",a,b);
    intercambio(&a,&b); /* llamada */
    printf("Despues de la funcion intercambio los valores son a=%d y b=%d\n",a,b);
    system("pause");
}

void intercambio (int *x,int *y)
{
    int aux;
    aux=*x;
    *x=*y;
    *y=aux;
    printf("En la funcion los valores son  a=%d y b=%d\n",*x,*y);
}
```

En este caso se utiliza el símbolo **&** (ampersand) delante de la variable enviada como parámetro, indicando así al compilador que la función que se ejecutará tendrá que obtener la dirección de memoria en que se encuentra la variable, logrando con esto el paso de parámetros por referencia.

Otro punto importante es que las variables usadas en la función **x, y**, son de tipo **puntero**, el único dato en C, que puede almacenar una dirección de memoria. Esto se especifica colocando el símbolo ***** (asterisco) delante de la variable.

Nota: Los punteros no son parte del contenido de este curso, por lo tanto solo se mencionan para ejemplificar el paso de parámetros por referencia.

4.3.2 RECONOCE LOS TIPOS DE VARIABLES EN TÉRMINOS DE SU ALCANCE.

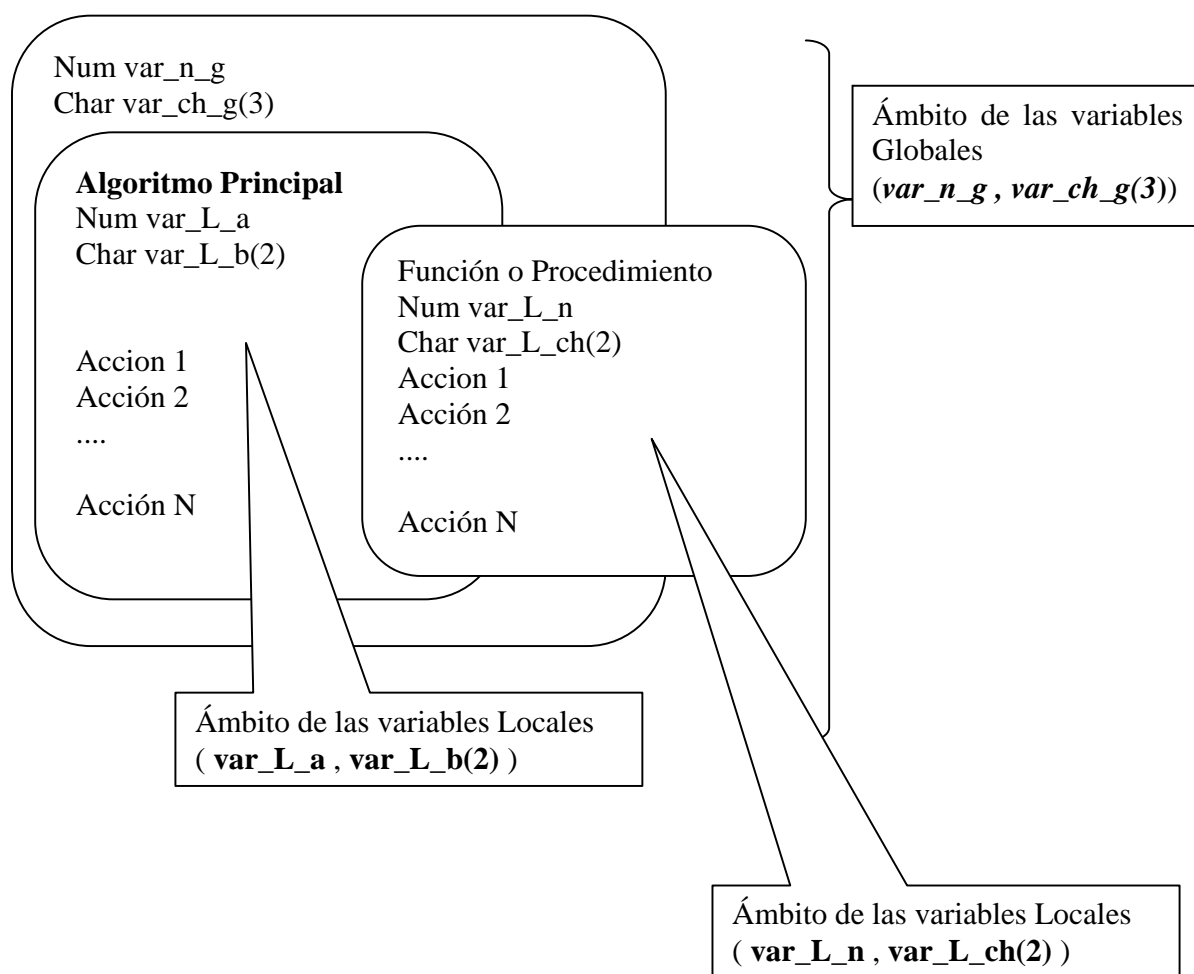
Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos:

- Variables Locales
- Variables Globales

Una **variable Local** es aquella que está declarada y definida dentro de un subprograma, en el sentido de que está dentro de ese subprograma y es distinta de las variables con el mismo

nombre declaradas en cualquier parte del programa principal. “El significado de una variable está confinado al procedimiento o función en la que está declarada”. Cuando otro subprograma utiliza el mismo nombre se refiere a una posición de memoria diferente.

Una **variable Global** es aquella declarada en el programa principal y fuera de todo Subalgoritmo (en muchos lenguajes de programación, incluso fuera del algoritmo principal) por lo que se asumirá para nuestro estudio que no existen variables de alcance global.



Reglas de ámbito:

- Un identificador se puede utilizar en el programa en el que está declarado y en todos los subprogramas de él.
- Si un identificador declarado en un programa **P** se redeclara en algún subprograma interno **Q**, entonces cualquier referencia a dicho identificador en **Q** (o algún subprograma de **Q**) utilizará el identificador declarado en **Q** y no el declarado en **P**. (Prevalece el más interno).

4.3.3 ANALIZA LAS VENTAJAS Y DESVENTAJAS DE LA MODULARIZACIÓN DE UN PROBLEMA

En virtud de los conceptos mencionados en los anteriores ítems, es fácil detectar las ventajas de la modularización destacando de entre ellas las siguientes:

- Flexibilidad y potencia para mejorar la productividad de un programa.
- Independencia y reusabilidad de los módulos dado que por sus características de unicidad de tareas, es fácil tomar un conjunto de instrucciones (que cumplen una función específica) y agregarlo a cualquier programa que se requiera.
- Estandarización de subproblemas.
- División de tareas entre varios programadores que permite la concentración de cada uno en problemas puntuales y de fácil resolución con lo que se puede optimizar el tiempo de desarrollo y escritura de algoritmos.

Sin embargo existen también desventajas que es necesario tomar en consideración y que dependen exclusivamente de la forma en que el programador utilice este método, esto se refiere a los denominados “Efectos laterales”.

Los **Efectos Laterales** responden a la manera de manipular las variables y su respectivo alcance dadas las características de las funciones y procedimientos en términos de los alcances de las variables globales y locales que se utilizan en el programa. *La comunicación entre un programa y un subprograma debe realizarse a través de parámetros, y no de variables globales.*

4.3.4 REALIZA PROGRAMAS EN C, USANDO FUNCIONES

EJERCICIOS PROPUESTOS:

1.- Construya un algoritmo que realice el cálculo de área y el perímetro de una circunferencia a través del diseño de funciones (una para calcular perímetro y otra para calcular área).

Area = $PI * R^2$ Perímetro = $2 * PI * R$.

Su algoritmo debe permitir ingresar los datos e imprimir en el programa principal.

2.- Realice un programa que permita imprimir la sumatoria de 10 números pares al cuadrado.

Considere:

- Su algoritmo debe permitir ingresar los datos e imprimir el resultado en el programa principal.
- Construya este programa solo con procedimiento.

3.- Realice un programa que calcule la función X, definida por:

$$X = (3^a + b^3 - 2) / (5ac)$$

Los valores de a, b y c son ingresados.

Su algoritmo debe permitir ingresar los datos e imprimir en el programa principal.

4.- Realice un programa que pueda leer X y luego pueda calcular la función Y definida por:

$$Y = (3X + 2) / (X - 2) * X$$

Considere:

- Que X no puede ser 0, porque no tiene solución la función. X es ingresado. Son 25 valores de X.
- Su algoritmo debe permitir ingresar los datos e imprimir en el programa principal.
- Construya este programa con función.

5.- Existe una lista de 10 alumnos con sus nombres y las notas de 2 pruebas y 2 controles los que tienen las siguientes ponderaciones, la prueba 1 (20%), prueba 2 (40%) y el promedio de controles (40%).

Construya un programa que lea los valores de las notas y luego imprima el promedio de cada alumno.

- Su algoritmo debe ingresar los datos e imprimir en el programa principal.
- Construya este programa con función.

OBJETIVO 5.1: REPRESENTAR ESTRUCTURAS DE DATOS ELEMENTALES

5.1.1: ESTRUCTURAS DE DATOS ELEMENTALES

Hasta ahora un dato implica una variable. El problema existe cuando tenemos una gran cantidad de datos relacionados entre sí. Para cada uno, una variable distinta.

Para dar solución a esta situación, se agrupan los datos en un solo conjunto, con un mismo nombre y se tratan como una sola unidad. Estos conjuntos reciben el nombre de estructura de datos.

Tipos de estructuras:

- ✓ Internas: residen en memoria. Por ejemplo : arreglos
- ✓ Externas: residen en dispositivos de almacenamiento. Por ejemplo: archivos

Estas estructuras de datos se encuentran prácticamente en la totalidad de los lenguajes de programación. Sin embargo en cada lenguaje se tratan e implementan de manera diferente.

En esta asignatura se revisará un tipo de estructura interna llamada Arreglos, y se implementará en lenguaje C.

5.1.2: CONCEPTO DE ARREGLO

Arreglo: Es una estructura de datos constituida por un número fijo de elementos, todos ellos del mismo tipo y ubicados en dirección de memoria físicamente contiguas.

Elementos de un arreglo: También denominados componentes, es cada uno de los datos que forman parte integrante del arreglo.

Nombre de un arreglo: Es la identificación utilizada para referenciar al arreglo y los elementos que la forman.

Tipo de datos de un arreglo: Marca del tipo básico que es común a todos y cada uno de los elementos o componentes que forman dicha estructura.

Índice: Es un valor numérico entero y positivo a través del cual podemos acceder directamente e individualmente a los distintos elementos que forman el arreglo, pues marca la posición relativa de cada elemento dentro del mismo. Aunque no todos los lenguajes operan de la misma forma, para efectos de lenguaje C, el primer elemento de un arreglo tiene índice 0.

Tamaño de un arreglo: Corresponde al número máximo de elementos que lo forman, siendo el tamaño mínimo un elemento y el tamaño máximo n elementos.

Acceso a los elementos o componentes de un arreglo: Los elementos de un arreglo tratados individualmente son auténticos datos básicos que reciben el mismo trato que cualquier otra variable. Para acceder o referenciar un elemento en particular es suficiente con indicar el nombre del arreglo seguida del índice correspondiente a dicho elemento, entre corchetes cuadrados [].

Dimensión de un arreglo: Determinado por el número de índices que necesitamos para acceder a cualquier elemento que lo forman.

Clasificación de los arreglos: Un arreglo se clasifica según sus dimensiones. Hay arreglos unidimensionales, bidimensionales y multidimensionales, en este curso se estudiarán los unidimensionales y los bidimensionales

5.1.3 : ARREGLOS UNIDIMENSIONALES O VECTORES

Arreglos Unidimensionales: También reciben el nombre de vector o lista y son estructuras de datos cuyos elementos son todos del mismo tipo y con las mismas características, y se referencia con un nombre común. Dichos elementos se ubican en posición de memoria físicamente contigua, poseen una sola fila y n columnas.

Ejemplo

Int Temp[6]

TEMP	5	100	25	34	0	3
Índice	0	1	2	3	4	5

Nombre del arreglo: **Temp**

Tipo de elementos : Numérico

Tamaño del arreglo: 6

Elementos del arreglo

En el elemento Temp[0] hay un 5

En el elemento Temp[1] hay un 100

En el elemento Temp[2] hay un 25

En el elemento Temp[3] hay un 34

En el elemento Temp[4] hay un 0

En el elemento Temp[5] hay un 3

Índices: 0,1, 2, 3, 4, 5

Dimensión: unidimensional, pues sólo requiere de un índice.

Un elemento cualquiera del arreglo se referencia mediante

nombre_arreglo[índice]

Por ejemplo:

```
Printf("%d", Temp[1]) ;    imprimirá 100
Printf("%d", Temp[5]);    imprimirá 3
Printf("%d", Temp[6]);    producirá un error
```

5.1.4: ARREGLOS BIDIMENSIONALES.

Una matriz, es un conjunto de datos homogéneos , para el lenguaje C, es un arreglo *multidimensional*. Se definen igual que los vectores excepto que se requiere un índice por cada dimensión.

Su sintaxis es la siguiente:

tipo nombre [tamaño 1][tamaño 2]...;

Una matriz *bidimensional* se podría representar gráficamente como una tabla con filas y columnas.

La matriz *tridimensional* se utiliza, por ejemplo, para trabajos gráficos con objetos **3D**.

En esta asignatura se trabajará con matrices o tablas de dos dimensiones, por lo tanto cada uno de los elementos que posee debe referenciarse por dos subíndices.

Ejemplo: Para una matriz de 4 x 5 o sea 4 filas y 5 columnas, tenemos:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

Se puede ir directamente a una posición determinada indicando sus coordenadas.

Ejemplo:

a[1][2]=5 Significa que la matriz “a” en la fila1, columna 2 almacena el valor 5.

OBJETIVO 5.2: CONSTRUYE ALGORITMOS Y PROGRAMAS EN C, UTILIZANDO PROCESOS TÍPICOS CON ARREGLOS.

5.2.1: DECLARACIÓN, INGRESO Y SALIDA DE DATOS EN ARREGLOS.

ARREGLOS UNIDIMENSIONALES:

La sintaxis de la declaración de un vector en C, es la siguiente:

Tipo_de_dato nombre_arreglo[tamaño]

Donde:

Tipo_de_dato : Indica el tipo de dato de cada uno de los elementos.
 nombre_arreglo : Cumple la misma norma que los nombre de variables.
 tamaño : Constante entera que indica el número máximo de elementos que tendrá el arreglo.

Ejemplo de declaración de arreglos unidimensionales

char nomb[30] : indica un arreglo de 30 elementos tipo caracter
 int edad[20] : indica un arreglo de 20 elementos enteros

NOTA: Los índices sólo pueden ser valores enteros. Los índices comienzan de 0. En este curso, el primer elemento de un arreglo tendrá índice 0. Al referenciar un elemento, si se sobrepasa el valor de índice se produce un error.

En lenguaje C, también es posible *inicializar* (asignarle valores) un vector en el momento de declararlo. Al realizar esto no es necesario indicar el tamaño. Su sintaxis es:

tipo nombre []={ valor 1, valor 2...}

Ejemplo 1:

int vector[]={1,2,3,4,5,6,7,8};

Gráficamente el vector quedaría:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Ejemplo 2:

char vector[]="programador";

char vector[]={ 'p','r','o','g','r','a','m','a','d','o','r' };

Gráficamente ambos vectores quedan:

p	r	o	G	r	a	m	a	d	o	r
---	---	---	---	---	---	---	---	---	---	---

IMPORTANTE:

1. Las listas o arreglos unidimensionales se declaran una sola vez.
2. Se debe distinguir claramente entre posición y contenido.
3. Se debe tener en cuenta que cuando se define un arreglo numérico, en cada elemento se podrán ingresar números, independiente de la cantidad de dígitos que tenga dicho número.

Ejemplo 1: El siguiente programa permite llenar un vector con los números enteros desde el 1 al 10, en este caso el ingreso de los datos al vector, se realiza asignando el contenido de una variable a cada posición del vector, luego muestra su contenido desde la primera posición a la última:

```
#include <stdio.h>
main()
/* Rellenar el vector y mostrar su contenido */
{
    int vector[10],i;
    for (i=0;i<10;i++) vector[i]=i+1;

    for (i=0;i<10;i++) printf(" %d",vector[i]);
}
```

Ejemplo 2: El siguiente programa permite crear y llenar un vector con números enteros ingresados por el usuario, luego muestra su contenido desde la última posición a la primera:

```
#include <stdio.h>
main()
/* ingresar datos al vector y mostrar su contenido */
{
    int vector[10],i;
    for (i=0;i<10;i++) scanf("%d", &vector[i]);

    for (i=9;i>=0;i--) printf(" %d",vector[i]);
}
```

ARREGLOS BIDIMENSIONALES

Para definir una matriz bidimensional:

Sintaxis: tipo nombre [tamaño 1][tamaño 2];

Donde:

Tamaño 1: representa la cantidad de filas.

Tamaño 2: representa la cantidad de columnas.

Al igual que en los vectores, los índices deben ser siempre valores enteros.

Si al declarar una matriz también queremos inicializarla, habrá que tener en cuenta el orden en el que los valores son asignados a los elementos de la matriz. por ejemplo:

```
int numeros[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

En este caso quedarían asignados de la siguiente manera:

```
numeros[0][0]=1  numeros[0][1]=2  numeros[0][2]=3  numeros[0][3]=4
numeros[1][0]=5  numeros[1][1]=6  numeros[1][2]=7  numeros[1][3]=8
numeros[2][0]=9  numeros[2][1]=10 numeros[2][2]=11 numeros[2][3]=12
```

Ejemplo: El siguiente programa permite crear una matriz de números enteros, con 4 filas y 5 columnas, rellenándola con datos ingresados por el usuario.

```
#include <stdio.h>

main() /* Rellenar y mostrar el contenido una matriz */
{
    int x,i,numeros[4][5];
    /* rellenamos la matriz */
    for (x=0;x<4;x++)
        for (i=0;i<5;i++)
            scanf("%d",&numeros[x][i]);
    /* visualizamos la matriz */
    for (x=0;x<4;x++)
        for (i=0;i<5;i++)
            printf("%d",numeros[x][i]);
}
```

Nota: en el ejemplo se puede ver que ahora es necesario contar con 2 ciclos, uno para recorrer las filas y el otro para las columnas. Dependiendo de cómo queremos ingresar o mostrar los datos de la matriz, avanza mas rápido el ciclo interno y mas lento el ciclo externo.

5.2.2 MANIPULACIÓN DE ARREGLOS

Variable auxiliar.

Las variables auxiliares son muy utilizadas para intercambiar los valores de dos variables sin perder sus contenidos o valores.

Para el caso de los arreglos se pueden usar para intercambiar elementos de posición, por ejemplo durante un proceso de clasificación u ordenamiento.

Se puede ejemplarizar mediante “tengo dos vasos, uno con agua y el otro con vinagre. Necesito intercambiar los líquidos”

Ejemplo. El siguiente trozo de código en C, permite el cambio de valor de dos variables

```
int a, b, aux;
a=5;
b=20;
Printf("El valor de a es =%d El valor de b es =%d", a,b);
aux=a;
a=b;
b=aux;
Printf("El valor de a es = %d El valor de b es =%d", a,b);
```

Recorrido de un arreglo:

Debido a que en la mayoría de los casos se sabe la cantidad de elementos de un arreglo, se recomienda el uso de la instrucción for, por su facilidad de uso.

Para recorrer un vector es necesario un ciclo, en cambio en una matriz se necesitan dos ciclos para poder pasar por todas las posiciones de ella.

Ejemplo 1:

El siguiente trozo de programa permite recorrer un vector y mostrar los elementos desde la posición 0 a la n-1, considerando n como el número de elementos del arreglo:

```
for (x=0;x<n;x++) printf("el elemento de la posición %d es %d",x,v[x]);
```

Ejemplo 2:

El siguiente trozo de programa permite recorrer un vector desde la posición n-1 a la 0, considerando n como el número de elementos del arreglo:

```
for (x=n-1;x>=0;x--) printf("el elemento de la posición %d es %d",x,v[x]);
```

Ejemplo 3:

El siguiente trozo de programa permite recorrer una matriz llamada Mat, de N filas por M columnas, por filas, es decir el ciclo que controla las filas avanza más lento:

```
for (x=0;x<N;x++)
    for (y=0;y<M;y++)
        printf("el elemento de la fila %d, columna %d es %d\n",x,y,Mat[x][y]);
```

Ejemplo 4:

El siguiente trozo de programa permite recorrer una matriz llamada Mat, de N filas por M columnas, por columnas, es decir el ciclo que controla las columnas avanza más lento:

```
for (y=0;y<M;y++)
    for (x=0;x<N;x++)
        printf("el elemento de la fila %d, columna %d es %d\n",x,y,Mat[x][y]);
```

5.2.3 PROCESOS TÍPICOS CON ARREGLOS

5.2.3.1 Operaciones sobre arreglos unidimensionales:

Como se ha visto hasta ahora el trabajo con vectores, se reduce a considerar la posición dentro del arreglo que ocupan sus elementos, y para ello se requiere usar un índice, cuyo valor debe ser entero y donde la primera posición corresponde al índice 0 (cero), para el lenguaje C, que será el utilizado para codificar los ejercicios.

Ejemplos sobre vectores:

1. Programa que permite llenar un arreglo de 100 elementos con números naturales del 1 al 100.

Solución

```
#include <stdio.h>
main()
{
    int i, A[100];
    for (i=0;i<100;i++) A[i]=i+1;
}
```

2. Dado un arreglo de 100 elementos numéricos enteros, generar el código que imprima la suma de todos los elementos del arreglo.

Solución

```
#include <stdio.h>
main()
{
    int i, A[100], suma=0;
    /*ingreso de datos al vector*/
    for (i=0;i<100;i++) scanf("%d",&A[i]);
    /*suma de sus elementos*/
    for (i=0;i<100;i++) suma+=A[i];
    printf("la suma de sus elementos es: %d\n",suma);
}
```

3. Dado un arreglo llamado “vector” lleno con 50 elementos numéricos enteros, generar un código que busque el mayor y menor elemento del arreglo.

Solución

```
#include <stdio.h>
main()
{
    int i, vector[50], min, max;
    /*ingreso de datos al vector*/
    for (i=0;i<50;i++) scanf("%d",&vector[i]);
    /*encontrar el mayor y menor*/
    min=vector[0];
    max=vector[0];
    for (i=0;i<50;i++)
    {
```

```

if (vector[i]>max) max=vector[i];
if (vector[i]<min) min=vector[i];
}
printf("El elemento mayor es: %d\n",max);
printf("El elemento menor es: %d\n",min);
}

```

4. Dado dos arreglos llamados “vector1” y “vector2” cada uno con 100 elementos enteros, genere un tercer vector llamado “resultado” que contenga la multiplicación de ambos arreglos.

Solución

```

#include <stdio.h>
main()
{
    int i, vector1[100],vector2[100],resultado[100];
    /*ingreso de datos a los vectores*/
    for (i=0;i<100;i++) scanf("%d",&vector1[i]);
    for (i=0;i<100;i++) scanf("%d",&vector2[i]);

    /*llenar el vector resultado*/
    for (i=0;i<100;i++) resultado[i]=vector1[i]*vector2[i];
}

```

5. Dado un arreglo de 100 elementos lleno con números enteros cualquiera, hallar el más pequeño e indicar su posición.

Solución

```

#include <stdio.h>
main()
{
    int i, n[100], indi=1;
    /*ingreso de datos al vector*/
    for (i=0;i<100;i++) scanf("%d",&n[i]);
    /*buscar el menor valor y su posición*/
    for (i=0;i<100;i++) if (n[i]<n[indi]) indi=i;
    printf("la posición del mínimo es: %d\n",indi);
    printf("su valor es : %d\n",n[indi]);
}

```

6. Dado un arreglo de 50 elementos enteros cualquiera, generar otro arreglo que contenga los elementos que están en las posiciones pares del primero.

Solución

```
#include <stdio.h>
main()
{
    int i, n[50], j=0, nuevo[25];
    /*ingreso de datos al vector*/
    for (i=0;i<50;i++) scanf("%d",&n[i]);
    /*llenar el segundo arreglo*/
    for (i=2;i<50;i=i+2)
    {
        nuevo[j]=n[i];
        j++;
    }
}
```

7. Dado un arreglo de 100 elementos numéricos enteros, dejar en la última posición el más grande.

Solución

```
#include <stdio.h>
main()
{
    int i, n[100], aux;
    /*ingreso de datos al vector*/
    for (i=0;i<100;i++) scanf("%d",&n[i]);
    /*recorrer el arreglo y dejar el elemento mayor al final*/
    for (i=0;i<100;i++)
        /*Si el número de la izquierda es más grande lo intercambiamos*/
        if (n[i]>n[i+1])
        {
            aux=n[i];
            n[i]=n[i+1];
            n[i+1]=aux;
        }
}
```

8. Dado un arreglo de 20 elementos, lleno con números enteros. Ingresar un número por teclado y buscar en el arreglo cuántos son mayores que él, cuántos son menores que él y donde se encuentra su igualdad.

Solución

```
#include <stdio.h>
main()
{
    int i, n[20], num, sup=0, inf=0;
    /*ingreso de datos al vector*/
    for (i=0;i<20;i++) scanf("%d",&n[i]);
    /*ingreso del valor a buscar*/
    printf("ingrese un valor: ");
    scanf("%d",&num);
    /*búsqueda del numero ingresado*/
    for (i=0;i<20;i++)
        if (num==n[i]) printf("Una igualdad se encontró en posición: %d",i);
        else if (num<n[i]) sup++;
        else inf++;
    printf("Existen %d numeros mayores a %d\n",sup, num);
    printf("Existen %d numeros menores a %d\n",inf, num);
}
```

9. Dado un arreglo A de 50 elementos enteros. Hacer un programa en C que permita llenar un arreglo numérico B de 5 elementos de tal forma que cada elemento del arreglo B tenga la suma de cada decena del arreglo A.

Solución

```
#include <stdio.h>
main()
{
    int i,A[30],B[3],h=0,x=10,acu=0,k=0;
    /*ingreso de datos al vector A*/
    for (i=0;i<30;i++) scanf("%d",&A[i]);

    /*llenado del vector B*/
    while (h<30)
    {
        for (i=h;i<x;i++) acu=acu + A[i];
        B[k]=acu;
        acu=0;
        k=k+1;
        x=x+10;
        h=i;
    }
}
```


5.2.3.2 Operaciones sobre Matrices:

En las matrices las operaciones de declaración, asignación, lectura y escritura obedecen a los mismos principios de los vectores, la diferencia es que como las matrices son arreglos bidimensionales requieren de dos índices [fila][columna] ; así pues tenemos que:

Ejemplos con matrices

1. Desarrollar un programa que permita llenar una tabla de 5 por 6 con números enteros introducidos por teclado. Imprimirla. Sumar todos los elementos de las columnas impares. Imprimir los elementos 2,3 y 3,2

Análisis

	<u>Elementos de la tabla</u>				
X	X	X	X	X	X
X	X	X	X	X	X
X	X	X	X	X	X
X	X	X	X	X	X
X	X	X	X	X	X

Las columnas impares son 1, 3 y 5. Los elementos de dichas columnas deben sumarse.

Solución

```
#include <stdio.h>
main()
{
    int mat[5][6], i, j, sum=0;
    /*para manejar matrices se deben declarar dos ciclos uno para cada índice*/
    for (i=0;i<5;i++)
        for (j=0;j<6;j++)
        {
            printf("ingrese el elementos de la posición %d,%d :",i,j);
            scanf("%d",&mat[i][j]);
            /*para determinar las columnas impares*/
            if ((j%2)==1) sum=sum + mat[i][j];
        }
    printf("Elementos de la tabla\n");
    printf("-----\n");
    /*nuevamente dos ciclos para imprimirla*/
    for (i=0;i<5;i++)
    {
        printf("\n");
        for (j=0;j<6;j++) printf("%d",mat[i][j]);
    }
    printf("\n La suma de los elementos de las columnas impares es: %d\n",sum);
    printf(" El elemento 2,3 es : %d\n",mat[2][3]);
    printf(" El elemento 3,2 es : %d\n",mat[3][2]);
}
```

2. Escriba un programa en C que permita almacenar en una matriz de nombre P el peso de 50 clientes de un gimnasio durante los doce meses del año pasado. Además, debe listar el promedio del peso anual de cada cliente.

La tabla tiene la siguiente representación gráfica:

1	2	3	4	5	12
74.3	70.5	67	65	65	...	59
80.6	80	80	78.6	78	...	78.2
64.5	64.5	64.5	60	60	...	60
70	71	71	70	73	...	70.2
...
58.5	56	55.5	55.5	50.3	...	49

Solución:

```
#include <stdio.h>
main()
{
    int P[50][12],i,k,suma;
    for (i=0;i<50;i++)
    {
        printf("Datos de Peso del cliente nro %d\n",i+1);
        for (k=0;k<12;k++)
        {
            printf("Peso del mes: %d ",k+1);
            scanf("%d",&P[i][k]);
        }
    }
    for(i=0;i<50;i++)
    {
        suma=0;
        for (k=0;k<12;k++) suma+= P[i][k];
        printf("El promedio anual del peso del cliente %d es: %d\n", i+1,suma/12);
    }
}
```

3. Generar una tabla de 4 por 5 con valores aleatorios entre 1 y 100. A partir de ella crear su transpuesta, es decir una matriz de 5 por 4.

La matriz transpuesta se forma poniendo las filas de la primera matriz como columnas en la segunda.

Análisis

A medida que vamos generando los elementos de la primera tabla se llena la segunda, es decir, la variable con la cual se controla la fila (i) de la primera matriz (a), corresponde a la columna de la segunda matriz (b); y la variable con la que se controla la columna (j) de la primera matriz (a), corresponde a la fila de la segunda matriz (b)

Solución

```
#include <stdio.h>
main()
{
    int a[4][5],b[5][4],i,j;
    for (i=0;i<4;i++)
        for (j=0;j<5;j++)
        {
            printf("ingrese un numero : ");
            scanf("%d",&a[i][j]);
            b[j][i]=a[i][j];
        }
}
```

5.2.3.3 Operaciones sobre Matrices y Vectores:

El siguiente ejemplo muestra el trabajo en conjunto de matrices y vectores:

1. Una empresa de publicidad tiene 100 empleados. Para efectos de remuneraciones, todos los empleados están diferenciados por categoría. Muchos empleados generan horas extraordinarias al mes y es necesario calcular el monto de cada uno considerando la cantidad de horas extraordinarias realizadas en un mes. Los antecedentes que la empresa nos proporciona son:
 - Existen 4 categorías de trabajadores.
 - Existe una matriz de 100 por 31 donde cada posición mantiene el número de horas extraordinarias trabajadas al día, durante un mes. También se dispone de dos vectores:
 - i. Uno de 100 elementos con la categoría que tiene cada trabajador
 - ii. Uno de 4 elementos con el valor de las horas extraordinarias por categoría.

Se pide hacer un programa que permita calcular:

- a) Cuanto le corresponde a cada empleado por concepto de horas extraordinarias durante el mes.
- b) Total pagado por la empresa por dicho concepto.

Análisis

En la matriz de 100 por 31, las horas realizadas por cada trabajador durante el mes están en una misma fila. Por lo tanto, se debe sumar toda una fila para saber la cantidad de horas extraordinarias trabajadas en el mes por un trabajador. Si esto se hace 100 veces, se obtiene la cantidad de horas extraordinarias generadas por cada trabajador durante el mes de 31 días.

En el vector de 100 elementos están las categorías de los trabajadores, el índice identifica a cada trabajador, vale decir este vector tiene solamente números entre 1 y 4 indicando la categoría del trabajador.

En la lista de categorías, hay valores que indican cuanto se paga por concepto de horas extraordinarias a cada categoría, vale decir, esta lista tiene cuatro cantidades. El índice indica la categoría.

Veamos un esquema

Matriz de 100 por 31, la llamaremos hor

0	1	2	3	4	30
3	5	1		4	...	6
2	1		4	1	4	3
1				4		2
		3	4	1	1	1
1	2	3				1
			1			

Ejemplo, el trabajador 2, tiene 1 hora extraordinaria el día 5, o sea, hor[1][4]=1

Vector de 100, lo llamaremos cat

0	1	2	3	4	99
3	3	2	1	2	...	4

El trabajador número 2 tiene categoría 3, vale decir, cat[1]=3

Vector de 4, lo llamaremos val

13456	18345	25230	28100
-------	-------	-------	-------

La categoría 3 tiene un monto asignado de \$25.230 pesos por cada hora extraordinaria

Solución

```
#include <stdio.h>
main()
{
    int hor[100][31],cat[100],val[4],i,j,tot=0,acuh=0;
    /*faltan los ingresos, se asume que están ingresados*/
    for (i=0;i<100;i++)
    {
        acuh=0;
        for (j=0;j<31;j++) acuh=acuh + hor[i][j];
        printf("El trabajador %d tiene horas extras %d A pago $ %d\n",i+1,acuh,acuh* val[cat[i]]);
        tot= tot + acuh*val[cat[i]];
    }
    printf(" Total a pagar por la empresa %d\n",tot);
}
```

5.2.4: ORDENAMIENTO Y BUSQUEDA EN VECTORES

5.2.4.1 Ordenamiento:

La ordenación consiste en reorganizar un conjunto de elementos de acuerdo a algún criterio determinado; estos criterios permiten agrupar elementos siguiendo una secuencia específica, permitiendo establecer una nueva ordenación.

Normalmente el orden es

Ascendente : de la a-z, A-Z, 0-9

Descendente : inverso a lo anterior

Si existen valores repetidos quedan en posiciones contiguas

Para llevar a cabo un ordenamiento existen muchos métodos y sirven tanto para arreglos numéricos como alfanuméricos.

Método de la burbuja

Se basa en llevar el máximo de la lista a la última posición. En la primera pasada se lleva el más grande al final. En las etapas siguientes se repite el proceso para llevar el segundo máximo a la penúltima posición y así sucesivamente.

La cantidad de pasadas es la cantidad de elementos –1

Ejemplo

Sea un arreglo numérico A de “n” elementos. Ordenar los elementos de menor a mayor.

Solución:

```
int A[n],i,k,aux;
for(k=0;k<n-1;k++)
    for (i=0;i<((n-1)-k);i++)
        if (A[i]>A[i+1])
        {
            aux=A[i];
            A[i]=A[i+1];
            A[i+1]=aux;
        }
```

Nota: Este trozo de código, corresponde a la implementación del método de ordenamiento de la burbuja, sin embargo, falta considerar el valor de n y el ingreso de datos al vector.

Método del switch

Es una variante del método la burbuja. También se basa en llevar el máximo al final del arreglo. Existe un switch que permite saber si la lista está ordenada o no.

Para el mismo ejercicio anterior

Solución.

```
int A[n],i,k=0,aux, sw=1;
while (sw!=0)
{
    sw=0;
    for (i=0;i<((n-1)-k);i++)
        if (A[i]>A[i+1])
        {
            aux=A[i];
            A[i]=A[i+1];
            A[i+1]=aux;
            sw=1;
        }
    k++;
}
```

Nota: Al igual que el trozo de código anterior, este solo corresponde a la implementación del método de ordenamiento. Falta considerar el valor de n y el ingreso de datos al vector.

Ejemplo aplicado:

Escriba un programa en C que permita que el usuario ingrese 100 números enteros para luego listar dicho números ordenados de menor a mayor.

Análisis

Asumamos el que el nombre del vector es Nor. Utilizaremos el método Burbuja.

Ejemplo gráfico:

Antes

Nor

10	23	35	14	25	68	72	205
1	2	3	4	5	6	7			100

Después

Nor

10	14	23	25	35	72	68	205
1	2	3	4	5	6	7			100

Solución:

```
#include <stdio.h>
main()
{
    int Nor[100],i,k,aux;
    printf("ingreso de datos al vector\n");
    for (i=0;i<100;i++) scanf("%d",&Nor[i]);
    /*ordenamiento*/
    for(k=0;k<99;k++)
        for (i=0;i<(99-k);i++)
            if (Nor[i]>Nor[i+1])
            {
                aux=Nor[i];
                Nor[i]=Nor[i+1];
                Nor[i+1]=aux;
            }
    printf("El vector ordenado es: \n");
    for (i=0;i<100;i++) printf("%d ",Nor[i]);
}
```

5.2.4.2 Búsqueda:

La operación de búsqueda consiste en determinar si un elemento determinado pertenece o no al conjunto de elementos que forman parte integrante de un arreglo y, en caso afirmativo indica la posición que dicho elemento ocupa.

Los métodos más usados de búsqueda son:

1. Búsqueda secuencial o lineal. (Veremos este)
2. Búsqueda binaria o dicotómica.

Nota: En esta oportunidad solo se revisará el método de búsqueda secuencial.

Búsqueda secuencial en un vector desordenado: Esta operación consiste en recorrer un vector secuencialmente de izquierda a derecha hasta encontrar el elemento buscado o hasta alcanzar el final del vector, en cuyo caso finalizará la operación de búsqueda sin haber localizado el elemento en cuestión. En aquellos casos en que el elemento buscado se encuentra repetido, indicará la posición del situado más a la izquierda, es decir, el de menor índice.

En el siguiente trozo de código se muestra solo lo correspondiente a la implementación del método de búsqueda secuencial. Vector es el nombre del arreglo numérico, lleno de números enteros y N es el largo del mismo.

```
int i=0,Vector[N],ele;
printf("Ingrese el elemento a buscar");
scanf("%d",&ele);
while ((i<N)&&(Vector[i]!=ele)) i=i+1;
if (i==N) printf("El elemento no se encuentra en el arreglo\n");
else
printf("El elemento se encuentra en la posición: %d \n",i);
```

Ejemplo

En un vector de nombre Cod se encuentra los 100 códigos de los productos de una bodega, en otro vector de nombre Pre se encuentran los precios de los 100 productos. Crear un programa en C, que al ingresar un código muestre el precio del producto si existe o, el mensaje “Producto no existe”.

Análisis

Asumamos los siguientes vectores como ejemplos para Cod y Pre

Cod

10	23	35	14	25	68	72	205
1	2	3	4	5	6	7			100

Pre

990	340	440	560	1020	1990	1100	1990
1	2	3	4	5	6	7			100

Solución:

```
#include <stdio.h>
main()
{
int j,i=0,Cod[100],Pre[100],ele;

/*ingreso de datos a los vectores*/

for (j=0;j<100;j++) scanf("%d", &Cod[j]);
for (j=0;j<100;j++) scanf("%d", &Pre[j]);
/*ingreso del elemento a buscar*/
printf("Ingrese el código del producto a buscar");
scanf("%d",&ele);
/*búsqueda*/
while ((i<100)&&(Cod[i]!=ele)) i=i+1;
if (i==100) printf("Producto no existe\n");
else
printf("El precio del producto %d, es: %d \n",Cod[i],Pre[i]);
}
```

Nota: Recuerde que los códigos fueron desarrollados en Dev C++, además falta incorporar la instrucción de pausa para poder visualizar correctamente los resultados.

Ejercicios propuestos de arreglos (OBETIVO 5.2)

1. Desarrollar un programa en C, que permita almacenar en un vector 50 elementos, números reales ingresados por el usuario. Después del almacenamiento debe mostrar el menor y mayor número que se encuentra en el arreglo.
2. Escribir un programa en C, que permita resolver el siguiente problema: En un centro meteorológico se han hecho en el día, N mediciones de temperatura. Al final de la jornada, se debe entregar cuál es la medición más alta y cuál es la más baja”.
3. Se tienen dos vectores A y B, con 20 elementos cada uno. Desarrolle un programa en C, que permita comparar cada elemento del vector A con el correspondiente del vector B, almacenando el elemento mayor entre ambos, en un tercer vector C. Si los elementos comparados son iguales, se traspasa cualquiera de ellos al vector C. Escriba el vector C.
4. Realizar un programa en C, que determine si el número 20 se encuentra en un vector llamado Nume de largo 5.000, lleno con números.
5. Realice un programa en C, que permita llenar un arreglo de 50 posiciones con números del 1 al 50. Luego mostrarlo por pantalla.
Mostrar el contenido de la posición 5, 15 y 33
6. Hacer un programa en C, que permita ingresar la edad de 350 personas a un vector llamado Edad y que genere otro vector que contenga las edades ordenadas de menor a mayor. Mostrar el contenido del vector ordenado.
7. Diseñar un programa en C, que permita crear una matriz de 10x10 de nombre TABLA, de tal manera que las filas que son pares se rellenan con 1 y las filas impares con 0. Una vez almacenados estos valores imprima su contenido en pantalla.
8. Desarrolle un algoritmo que genere la matriz unitaria. La matriz unitaria se forma con unos en la diagonal principal y ceros en el resto.
9. Desarrolle un algoritmo que permita llenar una matriz numérica con la multiplicación de los subíndices que la recorren más 6.
10. Desarrolle un algoritmo que permita Intercambiar la diagonal principal por la secundaria en una matriz de N x N.

ANEXO 1 – GLOSARIO

Algoritmo

Los algoritmos son procedimientos específicos que señalan paso a paso la solución de un problema y que garantizan el logro de una solución siempre y cuando sean relevantes al problema.

Dato

Es todo aquella representación de una entidad y que es susceptible de tratamiento ya sea en un programa o proceso informático. Por ejemplo nombre, apellido y edad son datos de una persona (entidad).

En otras palabras un dato es la representación de una realidad.

Información

Mensaje válido para un receptor o resultado del procesamiento de datos.

Computador

Máquina capaz de aceptar datos de entrada, procesarlos y entregar resultados de salida (información).

Programa

Conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema específico. Todo programa debe pasar por fases de elaboración hasta llegar al producto final.

Codificación

Consiste en la traducción del algoritmo a algún lenguaje de programación, el cual crea instrucciones entendibles y ejecutables por el computador; algunos de estos lenguajes pueden ser Pascal, C, C++, Visual Basic, Java, etc.

Dispositivos de Entrada

Como su nombre lo indica, sirven para introducir datos al computador para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Ejemplos: teclado, scanner, mouse, joystick, lápiz óptico, etc.

Unidad de Control

Coordina las actividades del computador y determina que operaciones se deben realizar y en que orden; así mismo controla todo el proceso del computador.

Unidad Aritmética - Lógica

Realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.

Dispositivos de Salida

Regresan los datos procesados que sirven de información al usuario. Ejemplo: monitor, impresora.

Memoria Central o interna

La CPU utiliza la memoria del computador para guardar información mientras trabaja. Mientras esta información permanece en memoria, el computador puede tener acceso a ella en forma directa. Esta memoria construida internamente se llama memoria de acceso aleatorio (RAM).

Memoria RAM (Random Access Memory)

Recibe también el nombre de memoria principal o memoria de usuario, en ella se almacena información sólo mientras el computador este encendido. Cuando se apaga o arranca nuevamente el computador, la información se pierde, por lo que se dice que la memoria RAM es una memoria volátil.

Memoria ROM (Read Only Memory)

Es una memoria estática que no puede cambiar, el computador puede leer los datos almacenados en la memoria ROM, pero no se pueden introducir datos en ella, o cambiar los datos que ahí se encuentran; por lo que se dice que esta memoria es de sólo lectura. Los datos de la memoria ROM están grabados en forma permanente y son introducidos por el fabricante del computador.

Memoria Auxiliar (Externa)

Es donde se almacenan todos los programas o datos que el usuario desee. Los dispositivos de almacenamiento o memorias auxiliares (externas o secundarias) más comúnmente utilizados son: cintas magnéticas, discos magnéticos discos ópticos.

Algoritmos cualitativos

Son aquellos en los que se describen los pasos utilizando palabras.

Algoritmos cuantitativos

Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

Lenguajes Algorítmicos

Es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.

Diagrama de flujo

Para el diseño de algoritmos se utilizan técnicas de representación. Una de estas técnicas son los Diagramas de Flujo (DDF), que se definen como la representación gráfica que mediante el uso de símbolos estándar unidos mediante líneas de flujo, muestran la secuencia lógica de las operaciones o acciones que debe realizar un computador, así como la corriente o flujo de datos en la resolución de problema.

Variable

Se considera variable a una zona de memoria referenciada por un nombre donde se puede almacenar el valor de un dato, que puede cambiarse cuando se desee. El nombre de la variable es elegido por el usuario pero debe seguir ciertas reglas. Una variable no es un dato, sino un área de memoria que contendrá un dato. A cada variable, el computador le asigna una dirección de memoria. Cuando se haga referencia a esa variable el computador irá siempre a esa dirección.

Constante

Es una posición de memoria, referenciada por un nombre, donde se almacena un valor que no puede cambiar y permanece invariable a lo largo del proceso. Tiene las mismas características de las variables.

Contador

Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción. Los contadores se utilizan en las estructuras de repetición con la finalidad de contar sucesos o acciones internas de bucle o ciclo.

Acumulador

Son variables cuyo valor se incrementa o decrementa en una cantidad variable, al igual que los contadores también necesitan inicializarse fuera del ciclo.

Concepto de Interruptor

Un interruptor o bandera (switch) es una variable que puede tomar los valores 1(verdadero) ó 0 (falso) a lo largo de la ejecución de un programa, dando así información de una parte a otra del mismo. Puede ser utilizado para el control de bucle.

Concepto de Bucle o ciclo

En informática, la mayoría de las veces las tareas que realiza el computador son repetitivas, lo único que varía son los valores de los datos con los que se está operando. Se llama bucle o ciclo a todo proceso que se repite un número de veces dentro de un programa.

Estructura de datos

Hasta ahora un dato implica una variable. El problema existe cuando tenemos una gran cantidad de datos relacionados entre sí. Para cada uno, una variable distinta.

Búsqueda

La operación de búsqueda consiste en determinar si un elemento determinado pertenece o no al conjunto de elementos que forman parte integrante de un arreglo y, en caso afirmativo indica la posición que dicho elemento ocupa.

Archivo

Definiremos archivo como un junto de datos que se encuentra físicamente en un soporte externo de la memoria, ya sea disco duro o disquete entre otros, de tal forma que los datos en él almacenados podamos utilizarlos o modificarlos. Corresponde a una estructura externa de datos. Un archivo esta compuesto por un conjunto de registros lógicos. Se referencian con un nombre y extensión. Por ejemplo: datos.txt

Registro

Corresponde a cada uno de los componentes del archivo. Un registro es una variable definida por el programador compuesto por un conjunto de variables que se referencian bajo un mismo nombre.

Campo

Un registro esta compuesto por unidades menores llamados campos, cada uno de los cuales pueden contener datos de diversos tipos, que tengan relación entre sí, porque todos hacen referencia a la misma y única entidad.

Archivos con organización secuencial

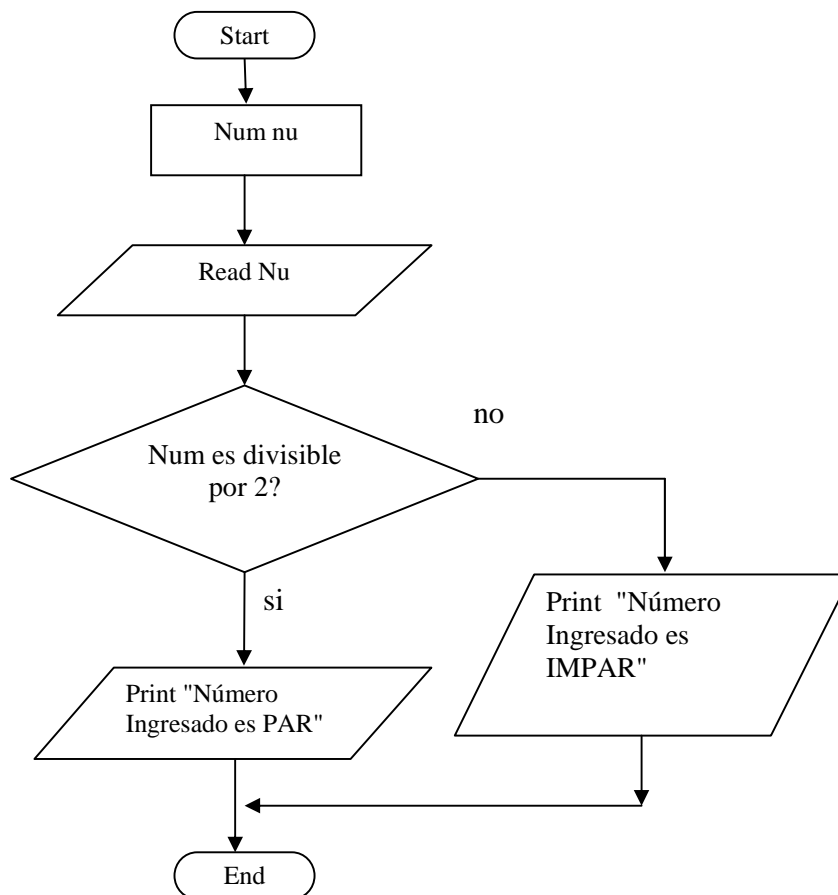
Los registros se escriben en orden correlativo en secuencia y se leen en el mismo orden que están escritos empezando por el primero. Para leer el registro N, deben leer todos los registros que ocupan las posiciones anteriores a N. No pueden existir espacios.

///

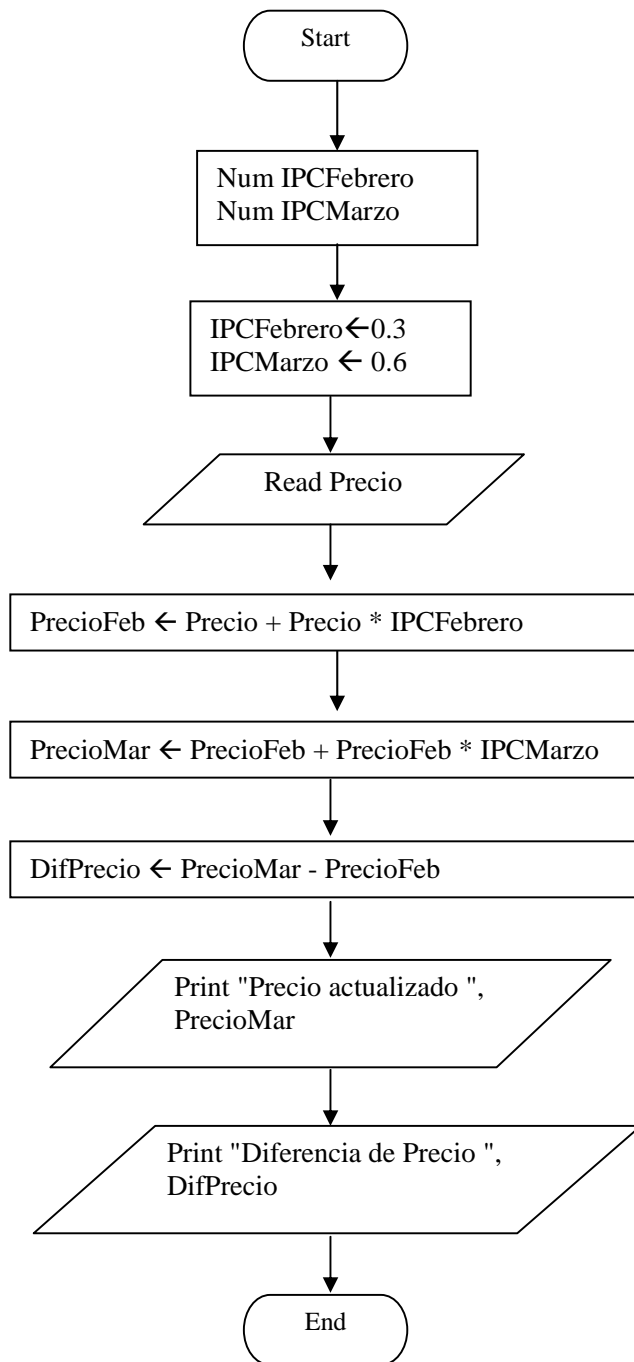
ANEXO 2 – SOLUCIÓN A EJERCICIOS PROPUESTOS

Solución OBJETIVO 2.4, Página 32

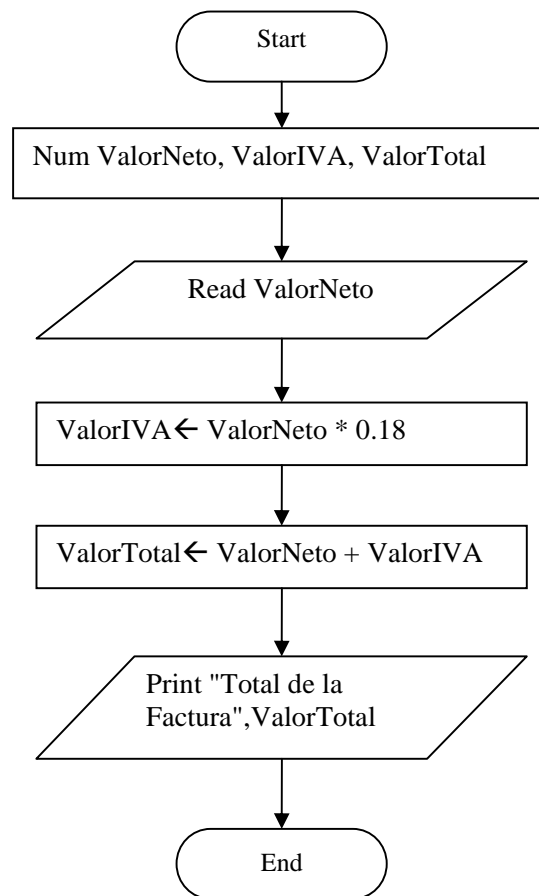
1.



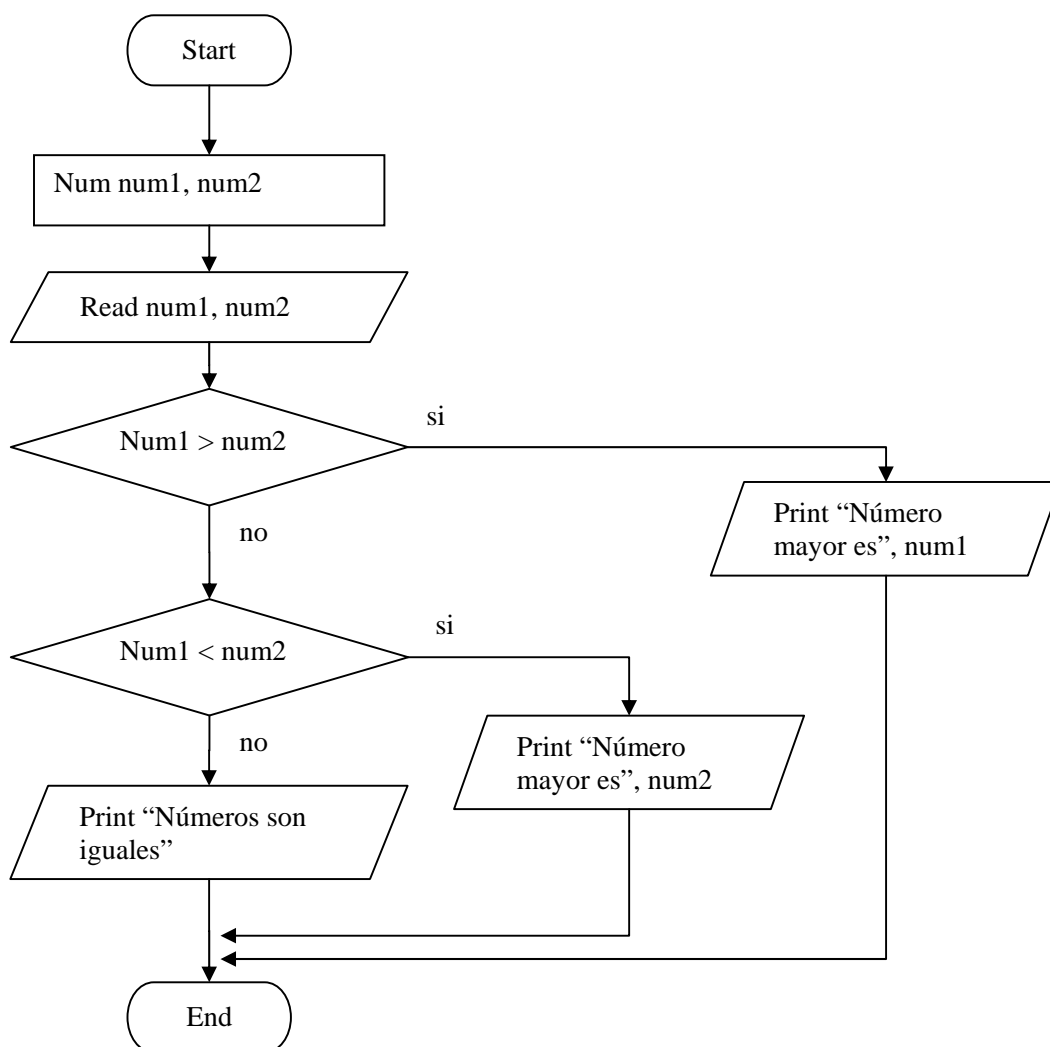
2.



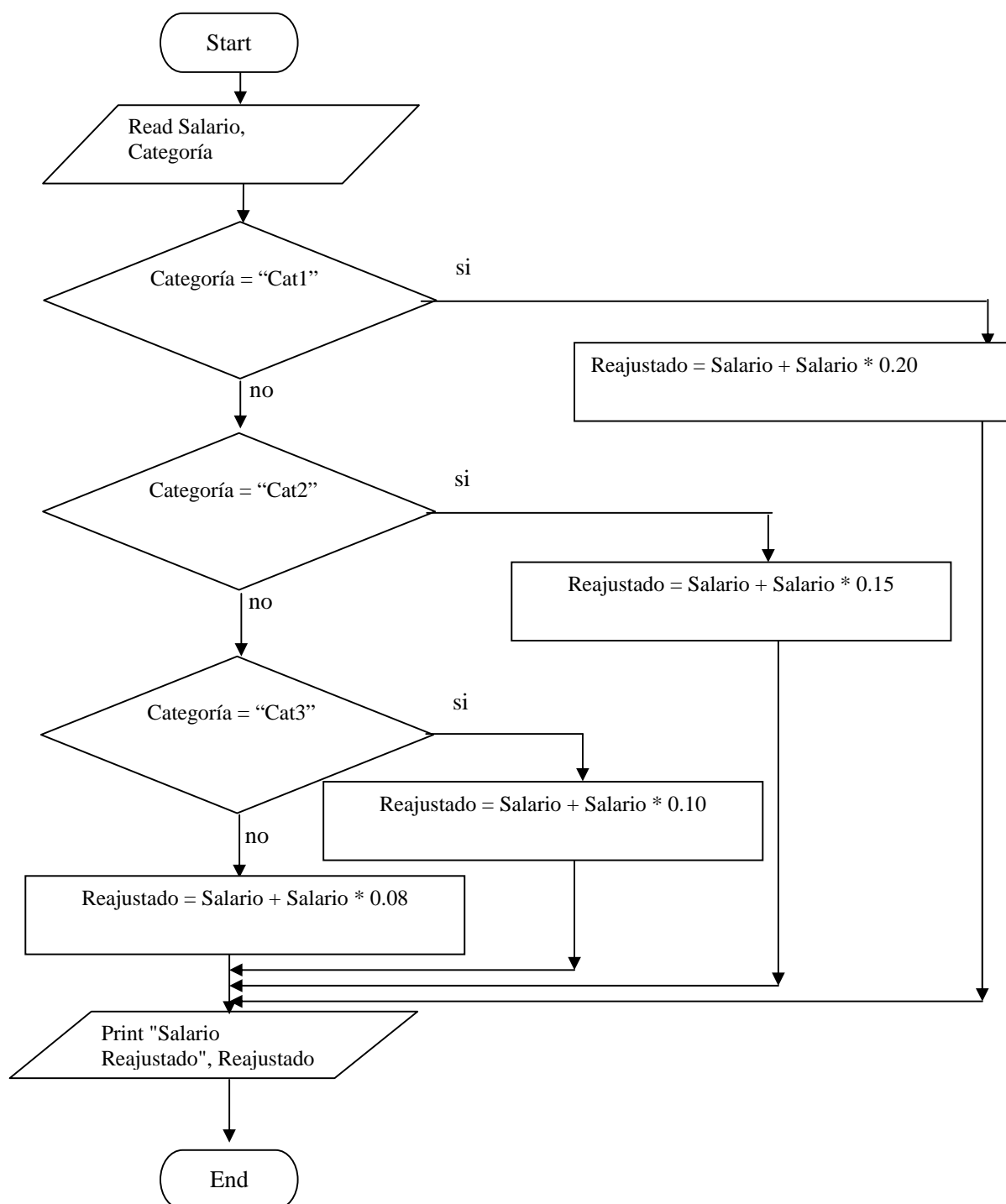
3.



4.



5.



6.

e	Salida
28	Adulto
15	Adolecente
21	Adulto
14	Infantil

Solución OBJETIVO 2.5, Página 43, 44

1. Indicar si son ciertas o falsas las expresiones:

- a) Falso
- b) Verdadero
- c) Verdadero
- d) Falso
- e) Verdadero
- f) Verdadero
- g) Verdadero
- h) Verdadero
- i) Falso
- j) Verdadero

2. Realice el siguiente cálculo respetando las prioridades

- a) 14
- b) 9.2
- c) 9.2
- d) 23
- e) 23
- f) 5.09
- g) 5.09
- h) 28.98

3. Transforme las siguientes expresiones algebraicas en expresiones algorítmicas utilizando los operadores informáticos:

- a) $M + N / (P - Q)$
- b) $M/N+4$
- c) $(M+N) / (P-Q)$
- d) $(P+N/P) / (Q-R/5)$
- e) $(3*x-y)/z + (2*x*y*y)/(z-1) + x/y$
- f) $a/(b-c/(d-(e/(f-g))))+(h+i)/(j+k)$

4. Resuelva las siguientes concatenaciones:

- a) "1212"
- b) "El resultado es"
- c) "Metodológico"
- d) "123.24-A"

5. Si $a = 10$, $b = 20$, $c = 30$, entonces

- a) Falso
- b) Verdadero
- c) Falso
- d) Verdadero

6. Calcular el valor de las siguientes expresiones aritméticas:
- a) 0
 - b) 5,5
 - c) 6
 - d) 3,5555
 - e) 11
 - f) 80
 - g) 16,333333
7. Calcular el resultado de las siguientes expresiones lógicas:
- a) Falso
 - b) Verdadero
 - c) Falso
 - d) Verdadero
 - e) Verdadero
 - f) Falso
8. Expresar el resultado de las siguientes expresiones:
- a) "PEDRO GOMEZ "
 - b) "GARCIA - GONZALEZ "
 - c) "CALLE- -MAYOR "
 - d) "12.465.450-k"
9. Escriba los siguientes intervalos numéricos en sus correspondientes intervalos lógicos:
- a) $x \geq 5$ and $x \leq 15$
 - b) $x > 120$ and $x \leq 200$
 - c) $x \geq 0$ and $x < 50$
 - d) $x > 15$ and $x < 30$
 - e) $x \leq 15$ and $x \geq 30$

Solución OBJETIVO 2.6.5, Página 53,54

1. **Start**
 num capital , resultado
 Read “Ingrese su capital: ”, capital
 resultado := (capital*2)/100
 Print “Su capital a fin de mes es de: ”, resultado
 End

2. **Start**
 #Solucion N°1
 num a,b := 0
 Print “Ingrese primer número: ”
 Read a
 Print “Ingrese segundo número: ”
 Read b
 If (a = b) Then
 Print “Los números son iguales: ”,
 Else
 If (a > b) Then
 Print “El número mayor es: ”, a
 Else
 Print “El número mayor es: ”, b
 End_if
 End_if
 End

 Start
 #Solucion N°2
 num a , b := 0
 Read “Ingrese primer número: ”, a
 Read “Ingrese segundo número: ”, b
 If (a = b) Then
 Print “Los números son iguales: ”, a
 Else
 If (a > b) Then
 Print “El número mayor es: ”, a
 Else
 Print “El número mayor es: ”, b
 End_if
 End_if
 End

3. **Start**
 num a , b , c
 Print “Ingrese tres números diferentes”
 Print “-----”
 Read “Ingrese primer número: ”, a
 Read “Ingrese segundo número: ”, b

```

Read "Ingrese tercer número: ", c
If ( a > b and a > c ) Then
    Print "El número mayor es: ", a
Else
    If ( b > a and b > c ) Then
        Print "El número mayor es: ", b
    Else
        Print "El número mayor es: ", c
    End_If
End_If

```

End

4. ***** propuesto *****

5. **Start**

```

num a , b
Read "Ingrese primer número: ", a
Read "Ingrese segundo número: ", b
If ( a = b ) Then
    Print "El resultado es: ", a*b
Else
    If ( a > b ) Then
        Print "El resultado es: ", a-b
    Else
        Print "El resultado es: ", a+b
    End_If
End_if

```

End

6. **Start**

```

# Sb ,variable numérica que significa sueldo base del empleado
# comisión, variable numérica que significa comisión por ventas del empleado
# SF ,variable numérica que significa sueldo empleado
num Sb,V1,V2,V3, Sf,comision
Print "INGRESE EL SUELDO DEL EMPLEADO"
Read Sb
Print "INGRESE PRIMERA VENTA DEL EMPLEADO"
Read V1
Print "INGRESE SEGUNDA VENTA DEL EMPLEADO"
Read V2
Print "INGRESE TERCERA VENTA DEL EMPLEADO"
Read V3
Comision := V1* 0.1 +V2*0.1+ V3*0.1
#Por 0.1 porque es el 10% o sea 10/100
SF := Sb + comision
Print "Monto de comisión", comision
Print "Sueldo final", SF

```

End

7. **Start**

```

# MC variable numérica que significa monto de compra
# DESC variable numérica que significa descuento de compra
# PF variable numérica que significa pago de compra
num SE, DESC, PF
Print "INGRESE MONTO DE COMPRA"
Read MC
DESC := MC * 0.15    # por 0.15 porque es el 15% o sea 15/100
PF := MC - DESC
Print "Monto de compra", PF

```

End
8. **Start**

```

# NP, PNP variable numérica que significa nota y porcentaje de promedio.
# EF, PEF variable numérica que significa nota y porcentaje de examen final
# TF, PTF variable numérica que significa nota y porcentaje trabajo final
num NP, PNP, EF, TF, PEF, PTF, CF
Print "INGRESE NOTA DE PROMEDIO"
Read NP
Print "INGRESE NOTA DE EXAMEN FINAL"
Read EF
Print "INGRESE NOTA TRABAJO FINAL"
Read TF
PNP := NP * 0.55      # por 0.55 porque es el 55% o sea 55/100
PEF := EF * 0.3       # por 0.3 porque es el 30% o sea 30/100
PTF := TF * 0.15      # por 0.15 porque es el 15% o sea 15/100
CF := PNP + PEF + PTF
Print "CALIFICACION FINAL", CF

```

End
9. **Start**

```

# CH, PH variable numérica que significa cantidad y porcentaje de hombres
# CM, PM variable numérica que significa cantidad y porcentaje de mujeres
# TE variable numérica que significa total de estudiantes
num CH, CM, TE, PH, PM
Print "INGRESE LA CANTIDAD DE HOMBRES"
Read CH
Print "INGRESE LA CANTIDAD DE MUJERES"
Read CM
TE := CH + CM
PH := ( CH * 100 ) / TE    # por regla de tres simple
PM := ( CM * 100 ) / TE    # por regla de tres simple
Print "PORCENTAJE DE HOMBRES", PH
Print "PORCENTAJE DE MUJERES", PM

```

End

10. **Start**

```

# montod  variable numérica que significa monto depositado
# percent  variable numérica que significa porcentaje de interés por el deposito
# interes1,interes2, los porcentajes obtenidos por los depósitos
#montof  variable numérica que significa monto final incluyendo intereses
num montod, percent, interes1,interes2,montof
Print "INGRESE MONTO DEPOSITADO"
Read montod
Print "INGRESE PORCENTAJE DE INTERES"
Read percent
Interes1 := (montod * percent)/100
If (interes1 > 7000) Then
    Interes2 := interes*percent
    montof := montod + interes1 + interes2
Else
    montof := montod + interes1
End_if
Print "monto final en su cuenta", montof

```

End
11. **Start**

```

Num Nota1,Nota2,Nota3,Promedio
Read "Nota 1: ", Nota1
Read "Nota 2: ", Nota2
Read "Nota 3: ", Nota3
Promedio:=(Nota1+Nota2+Nota3)/3
If (Promedio >=4.5) Then
    Print "Alumno aprobado"
Else
    Print "Alumno Reprobado"
End_If

```

End.
12. **Start**

```

num monto_venta, monto_pago
Read "Ingrese el monto de la venta:", monto_venta
If (monto_venta > 1000) Then
    Print "Presenta descuento"
    monto_pago:= monto_venta * 0.8
Else
    Print "No presenta descuento"
    monto_pago := monto_venta
End_If
Print "El monto a pagar es:", monto_pago

```

End.

13. **Start**

```
Num salario, horas, horas_extras
Read "Ingrese la cantidad de horas trabajadas:", horas
If(horas <=40) Then
    salario := horas * 16
Else
    horas_extras := horas - 40
    salario := 640 + (horas_extras*20)
End_If
```

End

14. **Start**

```
Num Conmul7, sumul7, conmul3, valor
Conmul7 := 0, Sumul7 := 0, Conmul3 := 0
Valor := 347
While (valor <= 2342)
    If (valor MOD 7 = 0 ) Then
        Conmul7 := conmul7 + 1
        Sumul7 := sumul7 + (valor^2)
    End_if
    If (valor MOD 3 = 0) Then
        Conmul3 ← conmul3 + 1
    End_if
    Valor := valor + 1
End_while
Print "La suma de los cuadrados de los múltiplos de 7 es: ", sumul7
Print "La cantidad de valores múltiplos de 7 es: ", conmul7
Print "La cantidad de valores múltiplos de 3 es: ", conmul3
```

End.

Start

```
Conmul7, sumul7, conmul3, valor
Conmul7 := 0, Sumul7 := 0, Conmul3 := 0
Valor := 347
For (valor := 347, valor <= 2342, valor := valor + 1)
    If (valor MOD 7 = 0 ) Then
        Conmul7:= conmul7 + 1
        Sumul7 := sumul7 + (valor * 2)
    End_if
    If (valor MOD 3 = 0) Then
        Conmul3 := conmul3 + 1
    End_if
    Valor := valor + 1
End_For
Print "La suma de los cuadrados de los múltiplos de 7 es: ", sumul7
Print "La cantidad de valores múltiplos de 7 es: ", conmul7
Print "La cantidad de valores múltiplos de 3 es: ", conmul3
```

End.

#Solución N°1

#Solución N°2

15. **Start**

```
Num Edad, sueldoact, sueldoreaj, contrab :=0
While (contrab <= n)
  Read edad, sueldoact
  If (edad < 26) Then
    Sueldoreaj := sueldoact * 1,10
  Else
    If (edad <= 35) Then
      Suledoreaj := sueldoact * 1,20
    Else
      If (edad <= 50) Then
        Sueldoreaj := sueldoact * 1,50
      Else
        Sueldoreaj := sueldoact * 1,70
      End_if
    End_if
  End_if
  Print "El sueldo reajustado es: ", sueldoreaj
  Contrabaj := contrabaj + 1
End_While
```

End.

Solución OBJETIVO 2.6.6, Página 64, 65, 66 y 67

1. Start

```

Num SumNota, i, Nota, Prom
SumNota := 0
i := 1
While (i<=7)
    Print "Ingrese Nota N°", i
    Read Nota
    SumNota := SumNota + Nota
    i := i + 1
End_While
Prom := SumNota/7
Print "Su promedio en la asignatura de Programación II es:", Prom

```

End

2. Start

```

Num i:=1, N
While ( i <= 10)
    Print "Ingrese un número"
    Read N
    If (N > 0) Then
        Print N
    End_If
    i := i + 1
End_While

```

End

3. Start

```

Num i, N, pos, neg, neu
i := 1, Pos := 0, Neg := 0, Neu := 0
While ( i <= 20)
    Print "Ingrese un número"
    Read N
    If (N = 0) Then
        Neu := Neu + 1
    Else
        If (N > 0) Then
            Pos := Pos + 1
        Else
            Neg := Neg + 1
        End_If
    End_If
    i := i + 1
End_While
Print "Cantidad de números Positivos: ", Pos
Print "Cantidad de números Negativos: ", Neg
Print "Cantidad de números Neutros: ", Neu

```

End

4. Start

```

num numero, contador
For (contador:=1, contador<=15, contador := contador +1 )
    # se lee el numero asegurando que sea negativo
    Do
        Read "Ingrese un número: ", numero
        While (numero >= 0)
            numero := ( -1 * numero )
        Print "El equivalente positivo es ", numero
    End_For
End

```

End

5. Start

```

num nota_actual, nota_promedio:=0, nota_baja:=7, contador
# inicialmente se supone que la nota más baja es un siete
For (contador:=1, contador<=40, contador := contador +1 )
    Read "Ingrese un nota ", nota_actual
    nota_promedio := nota_promedio + nota_actual
    # se esta usando nota_promedio como acumulador
    If ( nota_actual < nota_baja ) Then
        nota_baja := nota_actual
    End_If
End_For
nota_promedio := nota_promedio / 40
Print "El promedio de notas es ", nota_promedio
Print "La nota más baja es ", nota_baja

```

End

6. Start

```

num numero, máximo, producto, contador
# se leen los números asegurando que son positivos
Do
    Read "Ingrese la tabla a generar (multiplicando) ", numero
    While (numero < 0)
        Do
            Read "Ingrese el límite de generación (multiplicador) ", maximo
            While (maximo < 0)
                For (contador:=1, contador<=maximo, contador := contador +1 )
                    producto := ( numero * contador )
                    Print "El producto de ", numero, " por ", contador, " es ", producto
                End_For
            End_While
        End_Do
    End_While
End

```

End

7. Start

```

#variable hora para almacenar la hora
#variable min para almacenar los minutos
#variable seg para almacenar los segundos
num hora,min,seg
For(hora:= 0,hora<=23,hora := hora+1)
    For (min := 0,min <=59,min := min+1)
        For (seg := 0,seg<=59,seg := seg+1)

```

```

        Print  hora,":",min,":",seg
    End_For
End_For
End_For
End

```

8. Start

```

#montotal almacenará el monto total vendido por el vendedor
#comis almacenará el monto total de la comisión
#total almacenará el total de dinero que recibirá el vendedor
Num n,z,sbase,monto1,monto2,monto3,montotal,comis,total
Char nom
Print  "ingrese el número de vendedores:"
Read n
For(z:= 1 , z <= n , z := z+1)
    Print  "ingrese nombre vendedor:"
    Read nom
    Print  "ingrese sueldo base:"
    Read sbase
    Print  "ingrese monto venta 1:"
    Read  Monto1
    Print  "ingrese monto venta 2:"
    Read Monto2
    Print  "ingrese monto venta 3:"
    Read Monto3
    Montotal := monto1+monto2+monto3
    Comis :=montotal*10/100
    Total := sbase+comis
    Print  "Comisión ganada en la semana:" , Comis
    Print  "Total a pagar:" , Total
End_For
End

```

9. Start

```

Num ht,vh,he,sa
Char nom, Resp := "S"
While (resp = "S")
    Print  "ingrese nombre del obrero:"
    Read nom
    Read "ingrese horas trabajadas:"; ht
    If ( ht <=40 ) Then
        vh := 20
        sa := ht * vh
    Else
        he := ht - 40
        sa := 40*20+he*25
    End_If

```

```

        Print "el obrero " ; nom
        Print "Tiene un Salario semanal de : ", sa
        Print "Desea ingresar más datos(S / N)"
        Read Resp
        While (resp <> "S" or resp <> "N")
            Read Resp
        End_While
    End_While
End

10.    Start
        # Estadísticas de hombres y mujeres dentro de un grupo de N personas
        num ContH, ContM, i, N
        Char SEXO
        ContH := 0, ContM :=0, i :=1
        While (i <= N )
            Read SEXO
            If ( SEXO = "H") Then
                ContH :=ContH +1
            Else
                ContM :=ContM +1
            End_If
            i :=i +1
        End_while
        Print ContH, ContM
    End

11.    Start
        #promedio de calificaciones de un grupo de N alumnos
        num SumNotas:=0, i, NOTA, Prom, N
        Print " Cuantos alumnos : "
        Read N
        i:=1
        While ( i <= N )
            Read NOTA
            SumNotas :=SumNotas + NOTA
            i :=i +1
        End_While
        Prom:=SumNotas / N
        Print Prom
    End

12.    Start
        #Cálculo de Interés
        Num montodiner, Mes:=1
        Print " Ingrese Monto de dinero:"
        Read montodiner
        While ( Mes <= 12 )
            montodiner := montodiner + montodiner *0.02
            Mes :=Mes +1

```

```
End_While
Print "El monto al cabo de un año es", montodinero
```

End

13. Start

```
num ed, ph, pm, pt, she, sem, ch, cm
char resp, sex
seh:=0, sem:=0, ch:=0, cm:=0, resp:="s"
While ( resp = "s" )
    Read "Ingrese sexo del alumno (F/M) : ", sex
    Read "Ingrese edad del alumno: ", ed
    If (sex = "M") Then
        ch := ch + 1
        seh := seh + ed
    Else
        cm := cm + 1
        sem := sem + ed
    End_If
    Read "Hay más alumnos (s/n) ", resp
End_While
ph := seh/ch
pm := sem/cm
pt := (seh + sem) / (ch + cm)
```

End

14. Start

```
num n, numero, menor, mayor, contador
contador := 1
Read "Ingrese cantidad de números a procesar: ", n
Read "Ingrese numero: ", numero
mayor := numero
menor := numero
While ( contador < n )
    Read "Ingrese numero: ", numero
    contador := contador + 1
    If (numero < menor) Then
        menor := numero
    Else
        If (numero > mayor) Then
            mayor := numero
        End_If
    End_If
End_While
Print "El número mayor es : ", mayor
Print "El número menor es : ", menor
```

End

15. Start

```
num precio, totalcliente, totalcaja
```



```

Char resp1, resp2
totalcliente := 0, totalcaja := 0, resp1 := "s"
While ( resp1 = "s" )
    resp2 := "s"
    While ( resp2 = "s" )
        Read "Ingrese precio artículo: ", precio
        totalcliente := totalcliente + precio
        Read "Tiene mas artículos el cliente (s/n) ", resp2
    End_While
    Print "El total del cliente es : ", totalcliente
    totalcaja := totalcaja + totalcliente
    totalcliente := 0
    Read "Hay más clientes (s/n) ", resp1
End_While
Print "El total de la caja es : ", totalcaja
End

```

16. Start

```

#Programa Control de Pesos
num i,j,P,Acum,Peso,Prom_peso,Dif_peso
For (i:= 1,i <= 5, i := i+1)
    Print "Integrante contra la obesidad número ",i
    Print "Ingrese su peso desde la última reunión : "
    Read P
    # Solicitar el peso registrado por las diez básculas:
    Acum := 0
    For (j:=1, j<= 10, j:= j+1)
        Print "Ingrese peso bascula número",j
        Read Peso
        Acum := Acum + Peso
    End_for
    Prom_peso := Acum/10
    Dif_peso := Prom_peso - P
    If (Dif_peso > 0 ) Then
        Print "SUBIO DE PESO"
    Else
        If (Dif_peso < 0 ) Then
            Print "BAJO DE PESO"
        Else
            Print "SE MANTUVO EN EL PESO"
        End_if
    End_if
End_for
End

```

17. Start

#Hay que utilizar cuatro ciclos. Uno para el número de grupos (g), uno para el número de alumnos por grupo (n); uno para el número de materias por alumno

```
#y el último para ingresar las 3 notas por asignatura. Se debe imprimir el promedio
#de los grupos, el promedio de cada grupo y el promedio de cada alumno.
#El proceso de ingreso de notas se debe hacer por teclado y cada vez que
#se ingresan se acumulan para calcular el promedio de cada alumno, el que, a
#su vez se acumula para calcular el promedio de cada grupo y este se acumula
#para calcular el promedio de todos los grupos.
#Variables que se utilizarán
num i # contador para controlar el número de grupos.
num J # contador para controlar el número de alumnos por grupo.
num k # contador para controlar el número de materias por alumno.
num L # contador para controlar el número de notas por materia.
num Sumg # Acumulador donde se realiza la suma de los Promedios por grupo.
num Sumn # Acumulador donde se realiza la suma de los Promedios por alumno
num Summ # Acumulador donde se realiza la suma de las notas por alumno.
num Nota # Variable numérica para ingresar la nota
    I:=1, Sumg:=0
    While ( I <= g)
        J := 1, Sumn := 0
        While ( J <= n)
            K:=1, Summ:=0
            While ( K <= m)
                For ( L := 1 ; L <= 3; I := L + 1 )
                    Read "Ingrese nota" , nota
                    Summ := Summ + nota
                End For
                K:=K + 1
            End_While
            Summ := Summ/ (3*m)
            Print " EL Promedio del alumno es ", Summ
            Sumn :=Sumn + Summ
            J := J +1
        End_While
        Sumn :=Sumn/n
        Print " EL Promedio del grupo es ", Sumn
        Sumg := Sumg + Sumn
        i := i + 1
    End_While
    Sumg := Sumg / g
    Print " EL Promedio de los grupos es ", Sumg
End
```

18. Start

```
num Compra, Descuento, Total
char Color, Resp
Do
    Read "Ingrese Monto de la Compra", Compra
    Do
        Read "Ingrese color de la bolita", Color
        While (Color<>"roja" Or Color<>"amarilla" Or Color<>"blanca")
```

```

Case Color in
    "roja": Descuento:=Compra * 0.4
    Break
    "amarilla":Descuento:=Compra * 0.25
    Break
    "blanca":Descuento:=Compra * 0
End_Case
Total:=Compra – Descuento
Print "El descuento de su compra es de: ", Descuento
Print "El total a pagar es: ", Total
Read "¿Ingresa otro cliente? (S/N)", Resp
While (Resp <> "N")

```

End

19. Start

```

# A: Acumulador en donde se almacena el total del gasto de todos los artículos.
# P: Precio del artículo.
# C: Cantidad comprada de un artículo.
# D: Dinero gastado en un determinado artículo.
# resp: variable para determinar si continuamos agregando artículos al carrito.
num A, P, C, D
char resp
A := 0
Do
    Print "Ingrese el precio del artículo: "
    Read P
    Print "Ingrese la cantidad: "
    Read C
    D := P * C
    A := A + D
    Print "Dinero gastado en el artículo: ", D
    Print "Total de dinero gastado: ", A
    Print "Desea continuar S/N ?"
    Read resp
While ( resp = "S" )

```

End

20. Start

```

# P: Precio por asiento.
# C: Cantidad de asientos.
# E: Edad.
# C1: Acumulador de la Categoría 1
# C2: Acumulador de la Categoría 2
# C3: Acumulador de la Categoría 3
# C4: Acumulador de la Categoría 4
# C5: Acumulador de la Categoría 5
# resp: variable para determinar si continuamos
num P, C, E, C1:=0, C2:=0, C3:=0, C4:=0, C5:=0
char resp

```

```

Print "Ingrese el precio por asiento: "
Read P
Do
    Print "Ingrese la cantidad de Asientos: "
    Read C
    Print "Ingrese la Edad: "
    Read E
    If ( E < 5 ) Then
        Print "Los niños menores de 5 años no pueden entrar al teatro"
    Else
        If ( E < 15 ) Then
             $C1 \leftarrow C1 + C * P * 0.35$ 
        Else
            If ( E < 20 ) Then
                 $C2 \leftarrow C2 + C * P * 0.25$ 
            Else
                If ( E < 46 ) Then
                     $C3 \leftarrow C3 + C * P * 0.10$ 
                Else
                    If ( E < 66 ) Then
                         $C4 \leftarrow C4 + C * P * 0.25$ 
                    Else
                         $C5 \leftarrow C5 + C * P * 0.35$ 
                    End_If
                End_If
            End_If
        End_If
    End_If
    Print "Desea continuar S/N ?"
    Read resp
While ( resp = "S" )
    Print "Dinero que el teatro deja de percibir por categoría es la siguiente:"
    Print "Categoría 1: ", C1
    Print "Categoría 2: ", C2
    Print "Categoría 3: ", C3
    Print "Categoría 4: ", C4
    Print "Categoría 5: ", C5
End

```

21. Start

```

# Promedio: Promedio de la masa de aire de los neumáticos de n vehículos.
# Tipo: Tipo de vehículo 1 – motocicleta, 2 – automóvil.
# N: Número de vehículos.
# C: Contador de vehículos.
# A: Acumulador
# resp: variable para determinar si continuamos
# P: Presión
# V: Volumen
# T: Temperatura

```

```

num Promedio, Tipo, N, C:=1, A:=0, P, V, T
char resp
Print "Ingrese la cantidad de vehículos"
Read N
While (C <= N)
    Do
        Print "Ingrese el Tipo de vehículo 1 – motocicleta, 2 - automóvil:"
        Read Tipo
        While (Tipo<>1 and Tipo<>2)
            If ( Tipo = 1 ) Then
                Print "Rueda delantera"
                Read "Presión: ", P
                Read "Volumen: ", V
                Read "Temperatura: ", T
                 $M := (P * V) / (0.37 * (T + 460))$ 
                A := A + M
                Print "Rueda trasera"
                Read "Presión: ", P
                Read "Volumen: ", V
                Read "Temperatura: ", T
                 $M := (P * V) / (0.37 * (T + 460))$ 
                A := A + M
                C := C + 1
            End_If
            If ( Tipo = 2 ) Then
                Print "Rueda delantera izquierda"
                Read "Presión: ", P
                Read "Volumen: ", V
                Read "Temperatura: ", T
                 $M := (P * V) / (0.37 * (T + 460))$ 
                A := A + M
                Print "Rueda delantera derecha"
                Read "Presión: ", P
                Read "Volumen: ", V
                Read "Temperatura: ", T
                 $M := (P * V) / (0.37 * (T + 460))$ 
                A := A + M
                Print "Rueda trasera izquierda"
                Read "Presión: ", P
                Read "Volumen: ", V
                Read "Temperatura: ", T
                 $M := (P * V) / (0.37 * (T + 460))$ 
                A := A + M
                Print "Rueda trasera derecha"
                Read "Presión: ", P
                Read "Volumen: ", V
                Read "Temperatura: ", T
                 $M := (P * V) / (0.37 * (T + 460))$ 
                A := A + M
            End_If
        End_While
    End_Do
    C := C + 1
End_While

```

```

        C := C + 1
    End_If
End_While
Promedio:= A / C
Print "Promedio de masa del aire de los neumáticos de vehículos es: ", Promedio
End

```

22. Start

```

num limitetriple:=8, limitehorasextra:=40, i, n, horas, valorhora
num horasextra, horasextrasobre, sueldo,
Read "Ingrese número de trabajadores:", n
For(i:=1 , i<=n , i := i+1)
    Read "Ingrese horas trabajadas:", horas
    Read "ingrese el valor hora para el trabajador", valorhora
    Horasextra:=0
    Horasextrasobre:=0
    If (horas >40) Then
        Horasextra:=horas - 40
        If ( horasextra>8) Then
            Horasextresobre:=horasextra - 8
        End_If
    End_If
    sueldo:=(horas-horasextrasobre)*valorhora+(horasextra -
    horasextrasobre)*2*valorhora+(horasextrasobre)*3*valorhora
    Print "El sueldo del trabajador es:$", sueldo
End_For

```

End

23. Start

```

num n, cont, cont1, cont2, cont3, opcion
Cont := 0, cont1:=0, cont2:=0, cont3:=0
Print "Ingrese cantidad de diputados"
Read n
While (cont < n)
    Print "Ingrese su opción 1:A favor, 2:En contra, 3:Abstención"
    Read opcion
    Case opcion in
        1:    cont1 := cont1 + 1
              break
        2:    cont2 := cont2 + 1
              break
        3:    cont3 := cont3 + 1
    End_case
    Cont := cont + 1
End_While
Print " % diputados a favor", (cont1*100)/n
Print " % diputados en contra", (cont2*100)/n
Print " % diputados que se abstienen", (cont3*100)/n

```

End

24. Start

```

Char etiqueta, resp:="S"
Num precio :=0, total :=0, Valor_desc :=0
While (resp = "S")
    Print "Ingrese precio del producto"
    Read precio
    Print "Ingrese color de la etiqueta del producto"
    Read etiqueta
    If (etiqueta = "Rojo") Then
        Valor_desc := precio - (precio*20 / 100)
        Precio := Valor_desc
    End_if
    Total := total + precio
    Do
        Print " Desea continuar S/N ?"
        Read resp
        While (resp <> "N and resp <> "S")
    End_While
Print " El total a pagar será", total

```

End

25. Start

```

Char resp:="S"
Num Cont := 0, cont1 := 0, cont2 := 0, cont3 := 0, cont4 := 0, opcion
While (resp = "S")
    Print "Ingrese su opción 1:Estudios Básicos, 2:Estudios Medios"
    Print " 3:Estudios Superiores, 4:Postgrado"
    Read opcion
    Case opcion in
        1:    cont1 := cont1 + 1
              break
        2:    cont2 := cont2 + 1
              break
        3:    cont3 := cont3 + 1
              break
        4:    cont4 := cont4 + 1
    End_case
    Cont := cont + 1
    Do
        Print " Desea continuar S/N ?"
        Read resp
        While (resp <> "N and resp <> "S")
    End_While
Print " % Estudios Básicos", (cont1*100)/cont
Print " % Estudios Medios", (cont2*100)/cont
Print " % Estudios Superiores", (cont3*100)/cont
Print " % Estudios Postgrado", (cont4*100)/cont

```

End

26. **Start**
 Num OffSet, Estandar, Total_E:=0, Total_O:=0, Total
 Char Tipo
 Do
 Print "Ingrese tipo de solicitud(E=Estándar / O=Offset): "
 Read Tipo
 If (Tipo = "E") Then
 Print "Ingrese cantidad de pedido estándar: "
 Read Estandar
 Total_E :=Total_E + Estandar
 If (Total_E>50000) Then
 Total_E:=Total_E – Estandar
 Print "Solicitud rechazada"
 Else
 Print "Solicitud aceptada"
 End_If
 Else
 If (Tipo = "O") Then
 Print "Ingrese cantidad de pedido offset: "
 Read OffSet
 Total_O :=Total_O + OffSet
 If (Total_O>10000) Then
 Total_O :=Total_O - OffSet
 Print "Solicitud rechazada"
 Else
 Print "Solicitud aceptada"
 End_If
 End_If
 End_if
 Total:=Total_O+Total_E
 While (Total<=60000)
End
27. **Start**
 Num Cont:=0, Sum:=0, Sec
 Do
 Sec :=100 - (2*Cont)
 Sum := Sum + Sec
 Cont:=Cont +1
 While (Sec>0)
 Print Sum
End
28. **Start**
 Num Cont:=0, Nota, Cont_Rep:=0, Cont_Aprob, Porcen_R
 While (Cont<=50)
 Print "Ingrese Calificación"


```

Read Nota
If (Nota < 70) Then
    Cont_Rep := Cont_Rep + 1
End_If
Cont := Cont + 1
End_While
Porcen_R := ( Cont_Rep * 100 ) / 50
Print "Porcentaje de reprobados: ", Porcen_R

```

End

29. Start

```

Num N_Control, Sum:=0, Nota, Prom, Prom_May:=0, N_Control_May, i
Char Res:= "y"
Do
    Print "Ingrese Código de Control"
    Read N_Control
    For (i:=1, i<=5, i:=i+1)
        Print "Ingrese Nota de Unidad",i
        Read Nota
        Sum:=Sum+Nota
    End_For
    Prom:=Sum/5
    If (Prom>Prom_May) Then
        Prom_May := Prom
        N_Control_May := N_Control
    End_If
    Print "Desea leer otro alumno(s/n)"
    Read Res
While (Res<> "n")
Print "Código de Control alumno mayor promedio: ", N_Control_May
Print "Mayor Promedio: ", Prom_May

```

End

30. Start

```

Num Total_Alumnos:=0, i, j, Nota, Promedio, Sum_Nota
Char Alumno
For (i := 1, i<=40, i :=i+1)
    Sum_Nota:= 0
    Read Alumno
    For (J:= 1, j<=5, j:=j+1)
        Read Nota
        Sum_Nota := Sum_Nota + Nota
    End For
    Promedio :=Sum_Nota / 5
    If (Promedio < 3.0) Then
        Total_Alumnos ← Total_Alumnos + 1
    End_If
End_For
Print "Cantidad de alumnos sin derecho a nivelación : ", Total_Alumnos

```

End

31. Start

```

Num voto, Total_Votos:=0, Sum_Candi1:=0
Num Sum_Candi2:=0, Sum_Candi3:=0, i
Char Ganador
For ( i:= 1, i<= 250000, i:=i+1)
    Read voto
    If (voto = 1) Then
        Sum_Candi1 ← Sum_Candi1 + 1
    Else
        If (voto = 2) Then
            Sum_Candi2 ← Sum_Candi2 + 1
        Else
            Sum_Candi3 ← Sum_Candi3 + 1
        End_if
    End_If
End_For
If (Sum_Candi1 > Sum_Candi2) and (Sum_Candi1> Sum_Candi3) Then
    Ganador := “ Candidato 1”
    Total_Votos := Sum_Candi1
Else
    If (Sum_Candi2>Sum_Candi1) and (Sum_Candi2>Sum_Candi3) Then
        Ganador := “ Candidato 2 ”
        Total_Votos := Sum_Candi2
    Else
        Ganador := “ Candidato 3 ”
        Total_Votos := Sum_Candi3
    End_If
End_if
Print “ El Candidato Ganador es el ”, Ganador
Print “ Su Cantidad de Votos fue de ”, Total_Votos

```

End

32. Start

```

# La variable n representa el total de clientes
Num Suma_Clientes:= 0, i, n, T_ventas:=0, Compra := 0
For (i:=1, i<= n, i:=i+1)
    Read Compra
    T_ventas:=T_ventas + Compra
    Suma_Clientes :=Suma_Clientes + 1
End For
Print “ El total de Ventas del Día es” , T_ventas
Print “ El Número de clientes atendidos en el día es ”, Suma_Clientes

```

End

33. Start

```

# La variable n representa el total de clientes
Num i:=0, n, venta, paga, T_pagar:= 0, Iva:= 0, Cambio, T_Caja:= 0

```

```
While (i <= n)
    Print " Ingrese Venta"
    Read venta
    Iva := venta * 0.18
    T_pagar := venta + Iva
    T_Caja := T_caja + T_pagar
    Cambio := paga - T_pagar
    Print " El Iva del Cliente " , i , "es:", Iva
    Print " EL Total a pagar del Cliente" , i , "es:", T_pagar
    Print " El cambio del Cliente", i , "es:", Cambio
    i:=i+1
End While
Print "La cantidad de Dinero en Caja es :", T_caja
End
```

Solución OBJETIVO 3.2.6, Página 89

1.-

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

```
main()
{
    float f,c;
    printf("ingrese la temperatura en grados Fahrenheit : ");
    scanf("%f",&f);
    c=0.556*(f-32);
    printf("la temperatura en grados celsius es: %f\n",c);
}
```

2.-

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

```
main()
{
    float pi=3.14,radio;
    double area, volumen;
    printf("ingrese el radio de la esfera : ");
    scanf("%f",&radio);
    area=4*pi*pow(radio,3)/3;
    volumen=4*pi*pow(radio,3);
    printf("El area de la esfera es : %f\n",area);
    printf("el volumen de la esfera es : %f\n",volumen);
}
```

3.-

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

```
main()
{
    int x,y,pot=1,c=0;
    printf("ingrese la base :");
    scanf("%d",&x);
    do {
        printf("ingrese el exponente positivo: ");
        scanf("%d",&y);
    }
    while (y<0);
    while (c<y) {
        pot=pot*x;
        c++;
    }
    printf(" %d elevado a %d es %d\n",x,y,pot);
}
```

4.-

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

```
main()
```

```
{
```

```
int op,aa,an,edad,n,c=0,suma=0;
```

```
do
```

```
{
```

```
printf("Listado de opciones\n");
```

```
printf("1.- sumar 10 números enteros\n");
```

```
printf("2.- calcular edad aprox\n");
```

```
printf("3.- salir\n");
```

```
printf("ingrese el numero de la opcion:");
```

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

```
switch (op)
```

```
{
```

```
case 1: do
```

```
{
```

```
printf("ingrese valor nro %d: ",c+1);
```

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

```
suma+=n;
```

```
c++;
```

```
}
```

```
while (c<10);
```

```
printf("la suma de los 10 numeros es: %d\n",suma);
```

```
break;
```

```
case 2: printf("ingrese año de nacimiento: ");
```

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

```
printf("ingrese año actual : ");
```

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

```
edad=aa-an;
```

```
printf("la edad aprox. de la persona es %d\n",edad);
```

```
break;
```

```
case 3: printf("ud. eligio salir\n");
```

```
break;
```

```
default : printf("esa opcion no existe\n");
```

```
}
```

```
}
```

```
while (op!=3);
```

```
}
```

```
5.-
#include <stdio.h>
main()
{
float f,c;
printf("ingrese temperatura en grados Celsius:");
scanf("%f",&c);
f=c*1.8 + 32;
printf(" la temperatura en grados Farhenheit es: %f\n",f);
}
```

Solución OBJETIVO 4.3, Página 109

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

float area(float,float);
float perimetro(float,float);

main()
{
float a,p,pi=3.14,radio;
printf("Ingrese valor para el radio: ");
scanf("%f",&radio);
a=area(radio,pi);
p=perimetro(radio,pi);
printf("El area de la circunferencia es: %f\n",a);
printf("El perimetro de la circunferencia es: %f\n",p);
}

float area(float r,float pi)
{
float ar;
ar=pi*r*r;
return(ar);
}

float perimetro(float r,float pi)
{
float per;
per=2*pi*r;
return(per);
}
```

2.-

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

```
void sumatoria(int,int*);
```

```
main()
```

```
{
```

```
int n,c=0,suma=0;
```

```
do
```

```
{
```

```
do
```

```
{
```

```
printf("ingrese número par:");
```

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

```
}
```

```
while ((n%2)!=0);
```

```
sumatoria(n,&suma);
```

```
c++;
```

```
}
```

```
while (c<10);
```

```
printf("la sumatoria al cuadrado de 10 numeros es %d",suma);
```

```
}
```

```
void sumatoria(int numero, int *s)
```

```
{
```

```
*s=*s + (numero*numero);
```

```
}
```

3.-

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

```
#include <math.h>
```

```
double funcion_x(int, int, int);
```

```
main()
```

```
{
```

```
int a,b,c;
```

```
double x;
```

```
printf("ingrese valor de a :");
```

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

```
printf("ingrese valor de b :");
```

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

```
printf("ingrese valor de c :");
```

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

```
x=funcion_x(a,b,c);
```

```
printf("el resultado de la funcion es :%f",x);
```

```
}
```

```
double funcion_x(int a, int b, int c)
{
double res;
res=(pow(3,a)+b*3-2)/(5*a*c);
return(res);
}
```

4.-

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

```
int funcion_y(int);
```

```
main()
{
int x,y,c=0;
do
{
do
{
printf("ingrese un valor para x, mayor a cero:");
scanf("%d",&x);
}
while (x<=0 || x==2);
y=funcion_y(x);
printf("El resultado de la funcion es:%d\n",y);
c++;
}
while (c<25);
system("PAUSE");
}
```

```
int funcion_y(int x)
{
int res;
res=(3*x+2)/(x-2)*x;
return(res);
}
```

5.-

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

```
float promedio(float,float,float,float);
```

```
main()
{
float p1,p2,c1,c2,prom;
int c=1;
do
```



```
{
printf("ingrese las notas para el alumno Nro%d:\n",c);
printf("ingrese nota prueba 1:");
scanf("%f",&p1);
printf("ingrese nota prueba 2:");
scanf("%f",&p2);
printf("ingrese nota control 1:");
scanf("%f",&c1);
printf("ingrese nota control 2:");
scanf("%f",&c2);
prom=promedio(p1,p2,c1,c2);
printf("el promedio del alumno %d es %f\n",c,prom);
c++;
}
while (c<=10);
}

float promedio(float n1,float n2,float n3,float n4)
{
float p;
p=n1*0.2+n2*0.4+((n3+n4)/2)*0.4;
return(p);
}
```

Solución OBJETIVO 5.2, Página 129

1.-

```
#include <stdio.h>

main()
{
    int indice;
    float Vector[50],menor,mayor;
    for(indice=0;indice<50;indice++) scanf("%f",&Vector[indice]);
    mayor= Vector[0];
    menor= Vector[0];
    for (indice=1;indice<50;indice++)
    {
        if (Vector[indice]<menor) menor=Vector[indice];
        if (Vector[indice]>mayor) mayor=Vector[indice];
    }
    printf(" El número mayor encontrado en el arreglo es : %f\n",mayor);
    printf(" El número menor encontrado en el arreglo es : %f\n",menor);
}
```

2.-

```
#include <stdio.h>

main()
{
    int indice,N;
    float temp[N],menor,mayor;
    printf("Ingrese el numero de mediciones realizadas : ");
    scanf("%d",&N);
    for(indice=0;indice<N;indice++) scanf("%f",&temp[indice]);
    mayor= temp[0];
    menor= temp[0];
    for (indice=1;indice<N;indice++)
    {
        if (temp[indice]<menor) menor=temp[indice];
        if (temp[indice]>mayor) mayor=temp[indice];
    }
    printf(" La medicion mas alta es : %f\n",mayor);
    printf(" La medicion mas baja es : %f\n",menor);
}
```

3.-

```
#include <stdio.h>
main()
{
    int A[20],B[20],C[20],i;
    /*ingreso de datos a los vectores A y B*/
    printf("Ingrese valores al vector A:\n ");
    for (i=0;i<20;i++) scanf("%d",&A[i]);
    printf("Ingrese valores al vector B:\n ");
    for (i=0;i<20;i++)scanf("%d",&B[i]);
    /*llenado del vector C*/
    for (i=0;i<20;i++)
        if (A[i]>B[i]) C[i]=A[i];
        else C[i]=B[i];
    printf("Los elementos del vector C son:\n ");
    for(i=0;i<20;i++) printf("%d\n",C[i]);
}
```

4.-

```
#include <stdio.h>
main()
{
    int NUME[5000],cont=0,sw=0,i;
    /*ingreso de datos al vector NUME*/
    printf("Ingrese valores al vector NUME:\n ");
    for (i=0;i<5000;i++) scanf("%d",&NUME[i]);
    /*busqueda*/
    while ((cont<5000)&&(sw==0))
    {
        if (NUME[cont]==20) sw=1;
        else cont=cont+1;
    }
    if (sw==1) printf("El número 20 fue encontrado, en la posición %d\n",cont);
    else
        printf ("El número 20, NO fue encontrado \n");
}
```

5.-

```
#include <stdio.h>
main()
{
    int N[50],i;
    /*ingreso de datos al vector N*/
    printf("Ingrese valores al vector N:\n ");
    for (i=0;i<50;i++) scanf("%d",&N[i]);
    /*Mostrar el contenido del vector*/
    for (i=0;i<50;i++) printf("%d\n",N[i]);
    printf("El contenido de la posicion 5 es: %d\n",N[5]);
    printf("El contenido de la posicion 15 es: %d\n",N[15]);
}
```

```
printf("El contenido de la posicion 33 es: %d\n",N[33]);
}
```

6.-

```
#include <stdio.h>
main()
{
int edad[350],edad2[350],i,j,k,aux;
k=0;
for (i=0;i<350;i++)
{
    printf ("ingrese edad \n");
    scanf("%d",&edad[i]);
}
for (i=0;i<350;i++)
{
    for (j=i+1;j<350;j++)
        if (edad[i]>edad[j])
        {
            aux=edad[i];
            edad[i]=edad[j];
            edad[j]=aux;
        }
    edad2[i]=edad[i];
}
printf ("el vector ordenado es : \n ");
for (i=0;i<350;i++) printf ("%d",edad2[i]);
}
```

7.-

```
#include <stdio.h>

main()
{
int tabla[10][10],i,j;
for (i=0;i<10;i++)
    for (j=0;j<10;j++)
    {
        /*Asignar 1 a los pares y 0 a los Impares*/
        if ((i%2)==0) tabla[i][j]=1;
        else tabla[i][j]=0;
    }
/*se imprime la tabla*/
for (i=0;i<10;i++) for (j=0;j<10;j++) printf("%d\n",tabla[i][j]);
}
```

Nota: En este caso, producto de la operación % (Módulo), se consideró la fila 0 (cero) como número par, debido a que el resto de esa división da resultado 0.

8.- Se asumió una matriz de 10x10.

```
#include <stdio.h>
main()
{
int  tabla[10][10],i,j;
for (i=0;i<10;i++)
    for (j=0;j<10;j++)
    {
        /*Asignar 1 a la diagonal y 0 al resto*/
        if (i==j) tabla[i][j]=1;
        else tabla[i][j]=0;
    }
/*se imprime la tabla*/
for (i=0;i<10;i++)
{
for (j=0;j<10;j++) printf("%d",tabla[i][j]);
printf("\n");
}
}
```

9.-

```
#include <stdio.h>
main()
{
int  multi[10][10],i,j;
for (i=0;i<10;i++)
    for (j=0;j<10;j++) multi[i][j]=(i*j)+6;
/*se imprime la tabla*/
for (i=0;i<10;i++) for (j=0;j<10;j++) printf("%d\n",multi[i][j]);
}
```

10.-

```
#include <stdio.h>
main()
{
int A[10][10],i,j,k=9,aux;
for (i=0;i<10;i++)
{
    aux=A[i][i];
    A[i][i]= A[i][k];
    A[i][k]=aux;
    k=k-1;
}
/*se imprime la tabla modificada*/
for (i=0;i<10;i++) for (j=0;j<10;j++) printf("%d",A[i][j]);
}
```

Nota: en este caso, falta le ingreso de valores a la matriz A.