Guía teórica de lógica de programación.

UNEWEB

INDICE

- **Conceptos básicos sobre el computador** **4-6**
  - Computador u Ordenador 4
  - Datos 4
  - Hardware 4
  - Software 5
  - Sistema Operativo 5
  - Software de Aplicación 5
  - Software utilitario 6
  - Lenguajes de Programación 6
  - Diferencias entre los tipos de software 6

- **Estructura Funcional Del Computador (arquitectura de Von Neumann)** **7-9**
  - Concepto de programa almacenado 7
  - Memoria Principal(R.A.M.) 8
  - Unidad Central de Procesamiento (C.P.U.) 8
  - Unidad Aritmético Lógica (U.L.A.) 9
  - Dispositivos de Entrada/Salida 9
  - Buses o Unidades de Intercambio 9

- **Conceptos básicos sobre construcción de programas** **10-14**
  - Algoritmo 10
  - Pseudo-Codigo 10
  - Lenguaje de Programación 11
  - Programa 12
  - Programación 12
  - Aspectos que miden la calidad de los programas 12
  - Capacidad de abstracción 12
  - Faces en la resolución de problemas 13
  - Ciclo de vida de desarrollo de software y sus faces o etapas más usuales 14
    - Análisis 14
    - Diseño 14
    - Construcción 14
    - Compilación, Ejecución y verificación 14
    - Documentación 14
    - Depuración y mantenimiento 14

- **Conceptos Básicos** **15-20**
    - Dato 15
    - Tipo de Dato 15
    - Clasificaciones de los tipos de datos 15
        - Tipos de datos primitivos 15
        - Tipos de datos estructurados 15
        - Tipos de datos Abstractos 15
    - Variables 16
    - Constantes 17
    - Operaciones de los tipos de datos elementales 17
    - Expresiones 17
    - Prioridad de Operadores 18
    - Conversión de tipos 19
        - Conversiones de Ampliación 19
        - Conversiones de Restricción 19
        - Conversiones Implícitas 20
        - Conversiones Explicitas 20

- **Acciones Elementales** **21-23**
    - Operador de Asignación 21
    - Operación de Lectura Estándar 22
    - Operación de Escritura Estándar 22

- **Estructuras de Control de Flujo de Datos** **24-27**
    - Condicional 24-26
        - Condicional Simple 24
        - Condicional compuesto 25
        - Condicional Anidado 26
    - Selección Múltiple 27

- **Estructuras de Control De flujos de Datos Iterativas** **28-31**
    - For 29
    - While 30
    - Do . . . While 31

- **Principio de Programación Modular (Procedimientos)** **33-36**
  - Acciones 34
  - Funciones 35
  - Pase de Parámetros 36
    - Por Valor 36
    - Por Referencia 36

- **Ejercicios Propuestos** **37-44**
  - Tipos de Datos y Acciones Elementales 37-39
  - Estructuras de Control de Flujo De datos 39-40
  - Estructuras Iterativas 41-42
  - Programación Modular(Procedimientos) 43-44

Introducción a la Programación

Conceptos básicos sobre el computador

- **Computador u Ordenador**

Dispositivo electrónico utilizado para procesar datos en forma automática y obtener resultados los cuales pueden a su vez ser organizados y analizados para producir información. El computador está conformado por una parte física llamada **hardware** y por una parte lógica o de programas llamada **software.**

- **Datos**

Es una representación de hechos, conceptos o instrucciones, organizados de manera que se puedan procesar, interpretar o comunicar por medios humanos o automáticas. Los datos, son por ejemplo, representaciones de las características de una persona, objeto, concepto o hecho.

Los datos se pueden introducir en el computador como **entrada** y se procesan para producir **resultados** e información de **salida**.

- **Hardware**

En un computador se refiere a todos los componentes físicos que lo conforman, los aparatos propiamente dichos.

Como ejemplo tenemos los chips de los procesadores (CPU, procesadores matemáticos, procesadores de video), las tarjetas (la tarjeta madre, las tarjetas de memoria como la memoria RAM, las tarjetas de video, red, sonido), las unidades de almacenamiento (disco duro, disquete, cd, dvd, pen drive), los dispositivos periféricos (ratón, teclado, monitor, impresora)

- **Software**

Son los programas que permiten utilizar los recursos del computador. Programación, soporte lógico, parte no-mecánica o no-física de un sistema. Es un conjunto de programas y procedimientos que se incluyen en un computador o equipo con el fin de hacer posible el su uso eficaz de dicha máquina. Son las instrucciones responsables de que el hardware (la máquina) realice su tarea.

Como ejemplo tenemos los **sistemas operativos**, el **software de aplicación**, el **software utilitario** y los **lenguajes de programación.**

- **Sistema Operativo:**

Software básico encargado de controlar diferentes procesos en el computador mediante tres grandes funciones:

Coordinar y manipular el hardware del computador: como los procesadores, la memoria, las impresoras, las unidades de almacenamiento, el monitor, el teclado o el ratón;

Organizar los archivos en diversos dispositivos de almacenamiento, como discos flexibles, discos duros, cds;

Gestionar los errores de hardware y la pérdida de datos.

- **Software de Aplicación:**

Programa informático diseñado para facilitar al usuario la realización de un **determinado tipo** de trabajo. Algunas son aplicaciones desarrolladas **'a la medida'** que ofrecen una gran potencia y soluciones eficientes ya que están exclusivamente diseñadas para resolver un problema específico. Son ejemplos de este tipo de software los programas que realizan tareas concretas como manejo de nómina, análisis de estadísticas, manejo de almacén, etc.

- **Software Utilitario:**

Son programas que facilitan el uso del computador como herramienta para solucionar actividades generales como la edición de textos o la digitalización de materiales. En muchos casos los programas utilitarios son agrupados en **paquetes integrados** de software, por ejemplo el Microsoft Office o el OpenOffice, donde se ofrece soluciones más generales, pero se incluyen varias aplicaciones (procesador de textos, de hoja de cálculo, manejador de base de datos, correo electrónico, visor de imágenes, etc.).

- **Lenguajes de Programación:**

Sirven para escribir programas que permitan la comunicación usuario/máquina y las soluciones de problemas utilizando las ventajas, poder de cálculo, procesamiento y almacenamiento del computador.

- **Diferencias entre los tipos software**

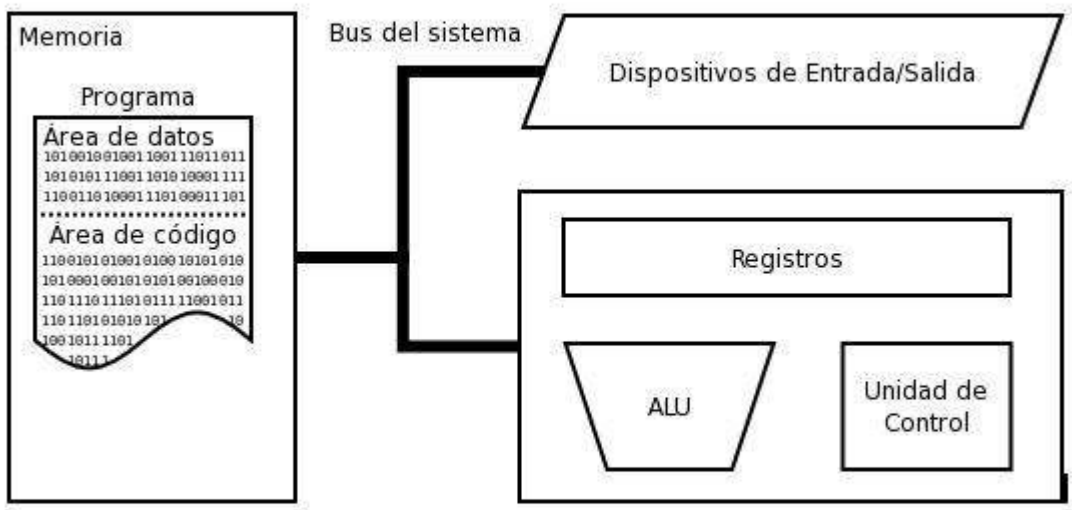
El **software de aplicación** se diferencia de un **sistema operativo** (que hace funcionar al ordenador), de una **utilidad** (que realiza tareas de mantenimiento o de uso general) y de un **lenguaje** (con el cual se crean los programas informáticos), en que suele resultar una solución informática para la automatización de tareas en un área determinada (procesamiento de texto, edición de imágenes, estadística, manejo de correspondencia, etc).

Estructura funcional del computador (Arquitectura de Von Neumann)

La **arquitectura Von Neumann** se refiere a las arquitecturas de computadoras que utilizan el mismo dispositivo de almacenamiento para las instrucciones y para los datos (a diferencia de la arquitectura Harvard). El término se acuñó en el año 1945, escrito por el conocido matemático John von Neumann, que propuso el concepto de **programa almacenado**.

Los ordenadores con arquitectura Von Neumann constan de las siguientes partes:

- ✓ La **Memoria Principal**,
- ✓ La **Unidad Central de Proceso** (CPU o Procesador), su vez formada por:
- ✓ La **Unidad Lógico-Aritmética** (ALU)
- ✓ La **Unidad de Control** (UC)
- ✓ Los **Dispositivo de Entrada/Salida**, formados por:
    - El **Bus de datos o unidades de intercambio** que proporciona un medio de transporte de los datos entre las distintas partes.
    - La **Memoria Secundaria,** Los **Dispositivos Periféricos**



- ✓ **Memoria Principal**

Es el subsistema donde se almacenan temporalmente los datos e instrucciones que son utilizados por el computador. Esta información está representada en una codificación binaria de 0 y 1. La memoria se divide en celdas, donde cada celda tiene una dirección única, de tal forma que el contenido de ellas puede ser buscado, extraído y utilizado.

- ✓ **Unidad Central de Proceso (CPU o Procesador)**

Es el subsistema encargado de extraer las instrucciones y los datos en la **memoria principal** para realizar los tratamientos u operaciones correspondientes. Está conformado por la **Unidad de Control** y la **Unidad Lógico-Aritmética**

La **Unidad de Control** (UC) se encarga de:

1. Obtener de la memoria la próxima instrucción a utilizar o ejecutar;

2. Decodificar la instrucción para determinar qué debe hacerse,

3. Según la decodificación hecha, enviar el comando apropiado a la ALU, memoria o controlador de entrada/salida para que realice la tarea

- ✓ La **Unidad Aritmético-Lógica (ALU)**

Se encarga de realizar las operaciones aritméticas y lógicas en él sistema necesarias para calcular direcciones de memoria, desplazamientos entre otras cosas.

- ✓ **Dispositivos de Entrada/Salida**

Es el subsistema que permite al computador interactuar con otros dispositivos, comunicarse con el mundo exterior y almacenar los datos y programas en unidades permanentes, por ejemplo, el disco duro.

Está conformado por:

- **Bus o unidades de intercambio**, que permiten comunicar información entre los componentes del sistema, los periféricos y el mundo exterior.
- **Memoria Secundaria**, permite conservar datos y programas en forma permanente, aún luego de apagar el computador.
- **Periféricos**, dispositivos utilizados para suministrar información entre el computador y el exterior (monitor, teclado, ratón, tarjeta de red, impresoras, tarjetas de memoria, escáner, etc.)

Conceptos básicos sobre construcción de programas

- ✓ **Algoritmo (*algorithm*)**

Es un conjunto bien definido de procedimientos lógicos o matemáticos que se pueden seguir para resolver un problema en un número finito de pasos.

Es una lista finita de pasos que plantean una solución a un problema, preferiblemente pasos los más cortos y simples posibles. Para un mismo problema pueden existir muchos algoritmos que conducen a su solución. La elección del mejor algoritmo está guiada por criterios de eficiencia y eficacia, entre otras características deseables.

Elementos de un algoritmo:

- Datos de entrada
- Proceso o pasos que resuelven el problema
- Datos de salida

- ➢ Características de un algoritmo:

Un algoritmo debe ser *preciso* e indicar el orden de realización de cada paso.

El resultado del algoritmo debe estar *definido*. Si se sigue un algoritmo dos veces con los mismos datos de entrada, se debe obtener el mismo resultado cada vez.

Un algoritmo debe ser *finito*. Si se sigue un algoritmo, se debe terminar en algún momento, es decir, se debe tener un número finito de pasos.

- ✓ **Pseudo-código (*pseudo-code*)**

En un algoritmo expresado de manera más formal. Se utiliza como una representación intermedia, antes de traducirlo o codificarlo con un lenguaje de programación. **En las clases de Lógica de Programación utilizaremos el pseudo-código para expresar las soluciones algorítmicas creadas. Recordemos que el objetivo de este curso es aprender a resolver problemas**

de una forma general, para que en los próximos cursos nos preocupemos solo en la sintaxis del lenguaje de programación que estemos aprendiendo.

- ✓ **Lenguaje de programación (*programming language*)**

En computación es cualquier lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones, a fin de que puedan ser procesadas por un computador.

Conjunto de caracteres, reglas, palabras y operaciones con significados previamente asignados y que permiten escribir programas.

La definición de un lenguaje de programación cubre tres aspectos:

1. Léxico: definen los símbolos que sirven para la redacción de un programa y las reglas para la formación de palabras en el lenguaje. Por ejemplo, 10 es un número entero

2. Sintaxis: conjunto de reglas que permiten organizar las palabras del lenguaje en frases, por ejemplo, la operación de división se define como Dividendo/Divisor

3. Semántica: definen las reglas que dan sentido a una frase

- Los principales tipos de lenguajes de programación utilizados en la actualidad son:

- ✓ Lenguajes de bajo nivel y traductores (lenguaje ensamblador, compiladores, intérpretes)
- ✓ Lenguajes de lato nivel (C++, C#, Visual Basic, Java, Turbo Pascal, Prolog, SQL, HTML, JavaScript, VBScript, PHP, VB.Net, Fortran, Delphi, etc.)

- **Programa (*program*)**

En Computación, es el conjunto de instrucciones que definen la secuencia de eventos que un computador debe ejecutar para llevar a cabo una tarea, realizando cálculos y suministrando resultados.

Grupo de instrucciones compuestas con la finalidad de resolver un problema específico mediante el uso de un computador. Un programa está codificado en un lenguaje que la máquina es capaz de entender y procesar.

Es la traducción de un **algoritmo** o de un **pseudo-código** utilizando un **lenguaje de programación**.

- **Programación (*programming*)**

Proceso que comprende el análisis del problema, diseño de la solución, escritura o desarrollo del programa, prueba del programa y su corrección.

Es la disciplina de la Computación que trata el desarrollo de programas.

- **Aspectos que miden la Calidad en un programa**

Algunos aspectos que se consideran para medir la calidad de un programa, también llamados **características deseables en un programa**, son:

- ✓ Legibilidad, Robustez, Eficacia
- ✓ Eficiencia, Adaptabilidad, Portabilidad
- ✓ Reusabilidad del software

- **Capacidad de Abstracción**

Mecanismo intelectual principal en la actividad de la programación, el cual durante la etapa de análisis del problema permite la separación de los aspectos relevantes de los irrelevantes en el contexto estudiado. Por ejemplo, si el problema consiste en determinar cuál es la persona más alta del salón, lo relevante es la estatura de cada persona, y no color de ojos, talla de calzado, etc.

- **Enfoques para solucionar un problema**

1. Programación **Modular**.

2. Enfoque **Divide y Vencerás**.

3. Diseño **Descendente (top-down)**. Ejemplo, para el problema de indicar los pasos para ver una película en el cine se podría considerar en un primer nivel los siguientes pasos: 1) Ir al cine, 2) Comprar una entrada, 3) Ver la película 4) Regresar a casa. Luego cada uno de estos pasos puede subdividirse en otros niveles donde las instrucciones sean cada vez más específicas.

- **Fases en la resolución de Problemas**

Resolver problemas a través de un computador se basa principalmente en analizar, diseñar, escribir y ejecutar un programa con pasos orientados a solucionar el problema. Podemos considerar como **fases de resolución de un problema** las siguientes:

1. Análisis del problema

2. Diseño del algoritmo, utilizando pseudo-código

3. Codificación, traducción de los algoritmos a un lenguaje de programación, esto nos permite crear el programa

4. Ejecución del código del programa

5. Verificación del programa

6. Documentación

7. Depuración de errores

8. Mantenimiento y mejora del programa

- **Ciclo de Vida de Desarrollo del Software y sus fases o etapas más usuales**

1. **Análisis**. El problema se analiza teniendo en cuanta los requerimientos o necesidades expresadas por el cliente, la empresa o las personas que utilizaran el programa.

2. **Diseño**. Una vez analizado el problema se diseña una solución que conduce a un algoritmo general que resuelve los elementos más significativos del programa. Este algoritmo puede ser escrito utilizando pseudocódigo.

3. **Construcción (implementación)**. La solución expresada en pseudocódigo se traduce a un programa que el computador pueda procesar utilizando un lenguaje de programación de alto nivel.

4. **Compilación, Ejecución y Verificación**. El programa se ejecuta (corre) y se verifica para eliminar errores de programación o de lógica.

5. **Documentación**. Se agrega al código del programa línea de texto que ayudan al programador y a las personas que a futuro harán mantenimiento al software a entender su estructura y comportamiento. La documentación también incluye escribir informes en donde se describe cómo se realizaron las diferentes fases del ciclo de vida del software (en especial los procesos de análisis, diseño, codificación y prueba), se agregan manuales de usuario y de referencia, así como normas para el mantenimiento.

6. **Depuración y Mantenimiento**. El programa se actualiza y modifica en la medida en que sea necesario de manera que cumpla con las necesidades de los usuarios las cuales también cambian en el tiempo.

Conceptos básicos

- **Dato**

Diferentes entidades u objetos de información con que trabaja un programa. Determina el conjunto de valores que la entidad puede almacenar, los operadores que puede usar y las operaciones definidos sobre ellos.

- **Tipo de Dato**

Define el conjunto de valores que un elemento o un objeto (una variable, constante, expresión o función) de dicho tipo puede asumir y las operaciones asociadas a tales valores.

Es un conjunto de entidades o de objetos y las operaciones definidas sobre ellos.

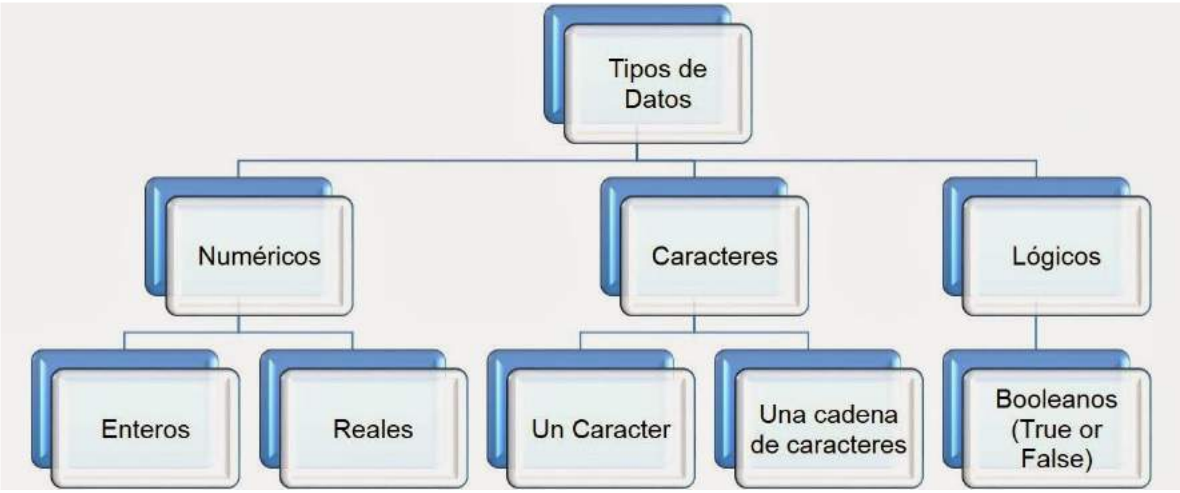
- **Ejemplos de Clasificaciones para los Tipos de datos :**

  - ✓ **Tipos de datos primitivos o elementales**: tipos básicos incluidos en cada lenguaje de programación, son aquellos que nos proporciona el lenguaje y con los que podemos (en ocasiones) construir tipos de datos estructurados y abstractos.

  - ✓ **Tipos de datos estructurados**: tipos basados o construidos a partir de tipos de datos primitivos (por ejemplo, arreglo, registro, archivo, conjunto).

  - ✓ **Tipos de datos abstractos** (TDA): tipos de datos definidos por el usuario y las operaciones abstractas aplicadas sobre ellos. Los TDA apoyan el concepto de ocultamiento de la información. Esconden los detalles de la representación y permiten el acceso a los objetos sólo a través de sus operaciones, son ejemplos las representaciones de los TDA Lista, Cola, Pila, Árbol y la representación que hace el Enfoque Orientado a Objeto mediante atributos y métodos.

En esta imagen se puede apreciar los tipos de datos elementales básicos en todo lenguaje de programación:



- **Variable**

Nombre asignado a una entidad que puede adquirir **un valor cualquiera** dentro de un conjunto de valores. Es decir, una entidad cuyo valor puede cambiar a lo largo del programa. En un programa de computador se puede asumir que una variable es una posición de memoria donde los valores asignados pueden **ser reemplazados o cambiados por otros valores** durante la ejecución del programa.

- **Constante**

Nombre asignado a una entidad al cual se asigna un valor que mantiene sin cambios durante el programa.

Operaciones de los tipos de datos elementales

Operación: Acción por medio de la cual se obtiene un resultado de un operando. Ejemplos: sumar, dividir, unir, restar.

Operando: número, texto, valor lógico, variable o constante sobre la cual es ejecutada una operación.

Operador: símbolo que indica la operación que se ha de efectuar con el operando, por ejemplo, + / - * > == ≠ ≥ =

- **Expresiones**

Son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones o acciones. Cada expresión toma un valor que se determina evaluando los valores de sus variables, constantes y operadores.

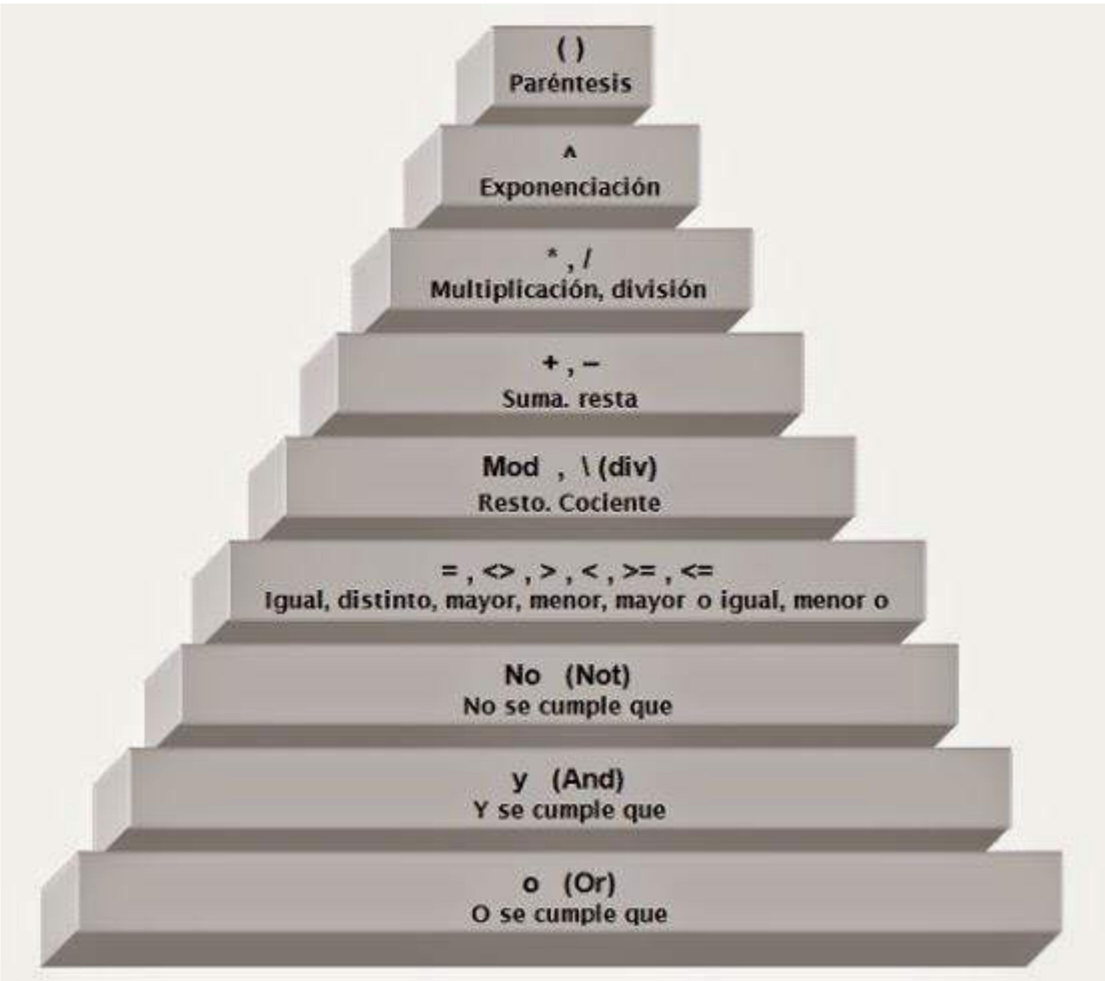
- ✓ Una expresión consta de *operandos* y *operadores*.
- ✓ Las expresiones se pueden clasificar en Aritméticas, Lógicas.

  Así tenemos:
  - Una expresión aritmética, arroja resultados de tipo numérico (*entero* o *real*)
  - Una expresión relacional o una expresión lógica, arrojan resultados de tipo lógico (*booleanos*)

- **Prioridad de Operadores**

Prioridad definida entre los operadores: Indica en qué orden debe aplicarse diferentes operaciones sobre un conjunto de valores. Permiten aplicar los operadores en el orden correcto.

De mayor prioridad a menor prioridad: En caso de haber operadores del mismo nivel en una expresión, se evalúan en orden de aparición de izquierda a derecha.



- **Conversión de tipos**

La conversión de tipos es el proceso de cambiar un valor de un tipo de dato a otro. Por ejemplo, el string o cadena "1579874" se puede convertir a un número entero, o se puede cambiar un número real a un String o a un Entero.

Las conversiones de tipo pueden ser de **ampliación** o de **restricción**:

- ✓ Las conversiones de **ampliación** transforman un valor de un tipo de dato a otro más grande, por ejemplo transforman un valor entero (conjunto más pequeño) a un número real (que es un conjunto más grande). Estas conversiones no producen desbordamiento o pérdida de datos.
- ✓ Las conversiones de **restricción** permiten transformar un valor de un conjunto de datos más grande a uno más pequeño, por ejemplo, transformar un número real a un entero o transformar una cadena a carácter. Si transformamos el valor X = 23,14587 a entero obtendríamos como resultado 23, lo cual significa pérdida de información ya que se pierde la precisión de los decimales; transformar el string m = ”casa” a carácter significaría quedarnos solo con el carácter „c‟ del inicio de la cadena. Estos tipos de transformaciones son poco convenientes ya que pueden implicar pérdida de información y solo deben ser usados cuando sea estrictamente necesario.

Las conversiones por ampliación o por restricción pueden a su vez ser **explícitas** o **implícitas**.

Las conversiones con pérdida de información tienen lugar cuando el tipo de datos original no tiene un análogo en el tipo de destino de la conversión. Por ejemplo, la cadena "Pedro" no se puede convertir en un número. En estos casos, algunos lenguajes de programación devuelven un valor predeterminado cuando se usa la función de conversión de tipo, por ejemplo el valor **NaN** o el número cero, indicando con estos valores que la conversión de tipos falló.

Algunos tipos de conversiones, como de una cadena a un número, tardan bastante tiempo. Cuantas menos conversiones utilice el programa, más eficaz será.

- **Conversión implícita**

La mayoría de las conversiones, como la asignación de un valor a una variable, se producen automáticamente. El tipo de datos de la variable determina el tipo de datos de destino de la conversión de expresión. En otros casos, la conversión implícita viene dada tanto por los tipos de datos como por los operadores utilizados.

- **Conversión explícita**

Para convertir explícitamente una expresión a un tipo de datos concreto, se utilizan funciones que se asumen predefinidas y disponibles en el pseudocódigo, colocando entre paréntesis el dato, variable o expresión que se va a convertir. Las conversiones explícitas requieren más escritura que las implícitas, pero proporcionan más seguridad con respecto a los resultados. Además, las conversiones explícitas pueden controlar conversiones con pérdida de información.

ACCIONES ELEMENTALES

- **Operador de Asignación**

Es el modo de darle un valor a una variable, el cual puede ser una constante, otra variable o el resultado de una expresión.

En pseudocódigo la operación de asignación se representa mediante el símbolo u operador **=** para la asignación.

Comportamiento: Modifica el estado de la variable.

La notación algorítmica (sintaxis) que utilizaremos para la asignación es:

<nombre de variable> = < constante o variable o expresión >;

Ejemplo: - Sean las siguientes variables:

char C1; int I; boolean Enc;

// a continuación se le asignaran valores a las variables

C1 = ‘a’; // a la variable C1 se le asigna o se le da el valor ‘a’

Enc = (1 > 500); // a la variable Enc se le asigna Falso, ya que no se cumple que 1 > 500

I = 5; // a la variable I se le asigna el valor 5

- **Operación de lectura y escritura estándar**

Lectura

La operación de **lectura o de entrada de datos** permite leer valores y asignarlos a constantes o a variables.

Almacena en una variable un valor extraído desde un dispositivo externo, del cual hacemos abstracción aunque generalmente es el teclado cuando se está programando. En pseudocódigo usamos la acción Leer para obtener los datos que nos suministra el usuario del algoritmo, datos necesarios para el procesamiento o cálculo posterior.

Los datos de entrada se introducen en el computador mediante dispositivos de entrada (teclado, pantalla, unidades de disco, escáneres, entre otros). Los datos de salida pueden aparecer en un dispositivo de salida (pantalla, impresora, cornetas, entre otros).

Comportamiento: La acción elemental Leer cambia el valor en la variable o variables que se está usando en la instrucción de lectura. Luego de leer un valor, el **valor de la variable cambia** en forma similar a si se hiciera una asignación.

La notación algorítmica (sintaxis) que utilizaremos para la asignación es:

read(<Nombre de variable>);

Escritura

Permite mostrar el valor de una variable, constante o expresión. Cuando estamos programando la acción Escribir trasmite un valor a un dispositivo externo, por ejemplo lo muestra por la pantalla del computador. En pseudocódigo la acción Escribir nos permite mostrar mensajes o mostrar los resultados al usuario del algoritmo.

Comportamiento: muestra mensajes y resultados almacenados en variables, constantes o expresiones.

La notación algorítmica (sintaxis) que utilizaremos para la asignación es:

write(< Nombre Variable, Constante o Expresión>);

O también

print(< Nombre Variable, Constante o Expresión>);

Estructuras de control de flujo de datos

- ✓ Condicional (if):
- ✓ Simple
- ✓ Compuesto
- ✓ Anidado

Simple: El condicional simple **Si – Entonces – Fsi** ejecuta un conjunto de instrucciones si cumple la condición evaluada en el Si.

Sintaxis:

If(<expresión lógica>){

<Bloque de instrucciones S1>

}

Comportamiento:

Si la condición es verdadera, ejecutan o realizan las instrucciones del bloque S1

Si la condición es falsa, no se realiza ninguna instrucción del Si

Recomendación: Cuando escribas las instrucciones en los algoritmos utiliza indentación o sangría para mostrar los niveles de anidación en las instrucciones. En este caso agregar un sangrado a la derecha para el bloque de acciones a fin de diferenciarlo de las instrucciones **Si - Entonces – Fsi**

Compuesto: Permite elegir entre dos opciones o alternativas posibles, en función de que se cumpla o no la condición expresada en el Si.

Sintaxis:

```
If(<expresión lógica>){
	<Bloque de instrucciones #1>
}else{
	<Bloque de instrucciones #2>
}
```

Comportamiento:

Si la condición es verdadera, ejecutan o realizan las instrucciones del bloque #1

Si la condición es falsa, se ejecutan o realizan las instrucciones del bloque #2

Recomendación: utilizar indentación o sangría en los algoritmos y códigos.

Anidado: Permite incluir dentro del cuerpo de la instrucción Si a otras instrucciones Si simples o compuestas. Se utiliza para elegir entre varias opciones o alternativas en función del cumplimiento o no de las diferentes condiciones que se van verificando en cada instrucción Si.

Sintaxis:

```
If(<expresión lógica #1>){

        <Bloque de instrucciones #1>

        If(expresión lógica #2){

                <Bloque de instrucciones #2>

        }else{

        <Bloque de instrucciones #3>

        }

}else{

        If(expresión lógica #3){

                <Bloque de instrucciones #4>

        }

        <Bloque de instrucciones #5>

}
```

Comportamiento: en cada condicional se cumple el mismo comportamiento que se ha indicado para el condicional simple o anidado.

Recomendación: utilizar indentación o sangría en los algoritmos y códigos. En este caso agregar sangría para diferenciar el nivel de las instrucciones de cada bloque SI**.**

}

Selección Multiple (Switch)

La selección múltiple permite evaluar una condición o expresión que puede tomar muchos valores distintos. Se ejecutaran las instrucciones correspondientes al caso que se cumple. La principal ventaja de la estructura Selección es que permite crear algoritmos que sean legibles y evitar la confusión creada por el anidamiento de muchos bloques SI.

Comportamiento: se evalúa cada condición de la selección y se realiza el bloque de instrucciones correspondientes a la condición que se cumple

Sintaxis:

```
Switch(variable){
        case (valor de variable: x):
                <Bloque de instrucciones >
        case (valor de variable: z):
                <Bloque de instrucciones >
           .
           .
           .
        case n:
                <Bloque de instrucciones >
        default:
                <Bloque de instrucciones>
}
```

Estructuras de control de flujo de datos iterativas

Un **bucle, lazo, ciclo o *loop*** (en inglés) es un segmento de algoritmo o programa (una secuencia de instrucciones) que se repiten un determinado número de veces mientras se cumple una determinada condición, en otras palabras, un bucle o ciclo es un conjunto de instrucciones que se repiten mientras una condición es verdadera o existe.

A cada repetición del conjunto de acciones se denomina **iteración**.

Para que un bucle no se repita indefinidamente debe tener una condición de **parada o de fin**. Esta **condición de parada** o de fin se verifica cada vez que se hace una iteración. El ciclo o *loop* llega a su fin cuando la condición de parada se hace verdadera.

La condición de parada puede estar al principio de la estructura repetitiva o al final.

Al igual que las estructuras de selección simple o compuesta (los bloques si – entonces – sino – fsi), en un algoritmo pueden utilizarse varios ciclos. Estos **ciclos pueden ser independientes** (uno a continuación de otro) **o anidados** (ciclos dentro de ciclos).

Para representar los bucles o lazos utilizaremos en el curso las estructuras de control:

- For(para)
- While(Mientras)
- Do…While (Repetir)

Vamos a utilizar ciclos cuando:

- Necesitemos **REPETIR INSTRUCCIONES** un determinado número de veces, mientras se cumpla una condición, mientras un hecho sea verdadero o hasta cuando se alcance un determinado valor o condición.

- Cuando necesitemos **CONTAR** un determinado número de elementos o de acciones, por ejemplo contar las sílabas de un texto, los elementos de una secuencia que verifican una determinada condición o contar el número de personas que cumplen ciertas características. En este caso se incluirán **contadores** dentro del bucle.

Los **contadores son variables** (generalmente de tipo **Entero**) que tienen un valor inicial y que se incrementan o decrementan en un valor constante cada vez que ocurre una iteración. Cuando los contadores se decrementan se habla de descontar, en lugar de contar.

-También usaremos ciclos cuando necesitemos **ACUMULAR** o **TOTALIZAR** terminados valores cada vez que se realiza una iteración. Los **acumuladores también son variables** (generalmente de tipo **Entero**, **Real** o **String**), que almacenan valores variables resultantes de las operaciones que se realizan en cada ciclo.

- **For**

Es una estructura iterativa que es controlada por una variable (llamada también **variable índice**), la cual se incrementa o decrementa hasta llegar a un valor límite o valor final que representa la condición de parada.

La estructura **for** comienza con un valor inicial de la variable índice, las acciones especificadas para el ciclo se ejecutan un número determinado de veces, a menos, que el valor inicial de la variable índice sea mayor que el valor límite que se quiere alcanzar.

SE RECOMIENDA USARLO: la estructura **for** es recomendada cuando se conoce el número de veces que se deben ejecutar las instrucciones del ciclo, es decir, en los casos en que el número de iteraciones es fijo y conocido.

El incremento o decremento de la variable_índice suele ser de 1 en 1, salvo cuando se indica lo contrario. La variable índice suele ser de tipo **Entero** y se utilizan comúnmente nombres como i, j o k (no importa si en mayúsculas o minúsculas)

Sintaxis:

```
For(int i=0; i<=10;i++){

        <Bloque de instrucciones s1>

}
```

En este ejemplo estamos creando una variable de tipo entero llamada i(por index, pero puede tener cualquier nombre) cuyo valor inicial será cero, la estructura repetirá el bloque de instrucciones s1 mientras la variable i cumpla la condición impuesta(que i sea menor o igual que diez).

El índice i se incrementará de uno en uno en cada iteración, lo que trae como resultado que el bloque de instrucciones s1 se ejecute 11 veces, desde cero hasta diez.

- **While**

Es una estructura iterativa que permite verificar la condición de **entrada** al ciclo **antes** del cuerpo de instrucciones a repetir.

Como la evaluación de la condición de entrada se realiza al **inicio** del bloque **while**, puede ocurrir que las instrucciones del ciclo no se realicen ni siquiera 1 vez, a diferencia del **do…while**, donde el bloque de instrucciones se realiza al menos 1 vez porque la condición de parada se verifica al final. Las instrucciones del while se pueden realizar 0 o más veces antes de que se cumpla la condición de terminar el ciclo.

El conjunto de instrucciones dentro while se ejecuta cuando la condición de entrada del principio se cumple (es verdadera). Dicho de otro modo, el ciclo de instrucciones dentro del **while** se va a detener cuando la condición se haga falsa.

Sintaxis:

While(<expresión lógica>){

        <Bloque de instrucciones>

}

SE RECOMIENDA USARLO: la estructura **while** es recomendada cuando tienes que verificar la condición de entrada al inicio y si se cumple, entonces, entrar al ciclo y realizar sus instrucciones.

Do… While

Ejecuta un bloque de instrucciones varias veces hasta que se cumple la condición que es verificada al final del bucle.

Las instrucciones dentro del ciclo **do…while** se van a realizar mientras la condición de parada evaluada al final sea verdadera. Dicho de otro modo, el ciclo se va a detener cuando la condición de parada se haga falsa

Sintaxis:

Do{

        <bloque de instrucciones>

}while(<expresión lógica>);

SE RECOMIENDA USARLO: la estructura **do…while** es recomendada cuando las instrucciones del ciclo se realizar **al menos 1 vez antes** de comprobar la condición de parada.

Procedimientos

La definición de procedimientos permite asociar un nombre a un bloque de instrucciones. Luego podemos usar ese nombre para indicar en algún punto de un algoritmo que vamos a utilizar ese bloque de instrucciones, pero sin tener la necesidad de repetirlas, sólo **invocando** al procedimiento por su nombre.

Los procedimientos pueden ser clasificados en acciones o funciones. Las **acciones** se caracterizan por **no retornar** valores al algoritmo que las llama, mientras que las **funciones retornan un** valor. Sin embargo, aunque las acciones no retornan valores, si pueden informar al algoritmo que las llamó de cambios realizados por sus instrucciones en algunos valores a través de una herramienta que se llama **pase de parámetros**. Los **parámetros** permiten utilizar la misma secuencia de instrucciones con diferentes datos de entrada. Utilizar parámetros es opcional.

Cuando entre las instrucciones de un algoritmo vemos el nombre de un procedimiento (acción o función), decimos que estamos **llamando o invocando** al procedimiento.

Los procedimientos facilitan la programación modular, es decir, tener bloques de instrucciones que escribimos una vez pero que podemos llamar y utilizar muchas veces en varios algoritmos. Una vez terminada la ejecución de un procedimiento (acción o función), se retorna el control al punto de algoritmo donde se hizo la llamada, para continuar sus instrucciones.

ACCIONES

Conjunto de instrucciones con un nombre que pueden ser llamadas a ejecución cuando sea necesario. No retornan valores.

Sintaxis:

```
void <nombre >(<lista de parametros>){

        <bloque de instrucciones>
}
```

Ejemplo

```
Void suma(int a,int b){
        Int res;
     res=a+b;
        write(“ el resltado es: ”+ res);
}
```

- **Funciones**

Al igual que las acciones con conjuntos de instrucciones con un nombre, pero se caracterizan por **retornar** (enviar o devolver) **un valor** al algoritmo que la llama.

Como el resultado de la función es retornado al algoritmo que la llamó, debe usarse una variable para almacenar este resultado, es decir, en una variable del algoritmo principal se "captura" el valor retornado por la función. Luego el valor almacenado en la variable puede ser utilizado por el algoritmo que llama a la función.

Sintaxis:

```
<tipo de dato de retorno><nombre>(lista de parámetros){
        <Bloque de instrucciones>
        return(valor a retornar);
}
```

Ejemplo:

```
int suma(int a, int b){
        int res;
        res=a+b;
return (res);
}
```

Con la instrucción return retornamos el valor al algoritmo que invoque la función suma.

Tipo de parámetros

- Parámetros actuales: Son los valores indicados en la llamada a la acción o función en el algoritmo principal. Son los valores que se desean pasar desde el algoritmo invocador a las instrucciones de la acción o función.
- Parámetros formales: Son los nombres dados a los parámetros en la definición formal de la acción o función. Con estos nombres se conocerán a los valores de los parámetros dentro de la acción o función y funcionan como variables locales dentro de ellos.

Pase de parámetros por valor

El parámetro actual no es modificado si se modifica el parámetro formal dentro del procedimiento, ya que **ambos parámetros ocupan posiciones diferentes en memoria**. Esto se debe a que el parámetro actual se evalúa y el resultado se copia en el correspondiente **parámetro formal, que ocupa otra posición en memoria**. Por ello, cuando se regresa de la acción o función al algoritmo desde donde se hace la llamada los parámetros actuales mantienen su valor original.

Pase de parámetros por referencia

El parámetro actual sufre los mismos cambios que el parámetro formal. El parámetro actual no puede ser ni una constante ni una expresión. **Ambos parámetros ocupan la misma posición en memoria**. Por ello, cuando se regresa de la acción/función al algoritmo desde donde se hace la llamada los parámetros actuales han cambiado.

Ejercicios Propuestos

Tipos de datos y acciones elementales

Ejercicio 1.
¿De cuál tipo de dato sería la variable donde almacena lo siguiente?

- "Hola Mundo"
- Verdadero
- π
- '1'
- 'c'
- 256
- 8>19

Ejercicio 2.
¿Siguiendo la prioridad de operadores, convierta a expresión matemática, resuelva e indique en cuál tipo de variable almacenará el resultado de las siguientes expresiones:

- (5 + 3 * 2) + 9 > 3 * 5 * 14 % 3
- 2 *(4 – 10 + 8)/2* 36 *(1/2)
- 260 / 12 + 54 % 3 – 85 % 7
- (48 < 2 * 3) | | (2 * 7 < 12)
- ((8 > 2) | | (932 < 23)) && 4 == 2

Ejercicio 3.
Escribir las siguientes expresiones matemáticas a pseudo-codigo.

- $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$
- $a^2 + b^2 = c^2$
- $(1 + x)^n = 1 + \frac{nx}{2} + \frac{nx^2}{3}$

Ejercicio 4.
Dados dos (2) números calcule la suma, resta, multiplicación, división y módulo.

Ejercicio 5.
Dados tres (3) números, Hacer una aplicación que calcule la resolvente.

Ejercicio 6.
Dados dos (2) lados de un triángulo en cm, calcular la hipotenusa del mismo.

Ejercicio 7.
Dado un (1) número, imprimir 0 si es par y 1 si es impar.
Ejercicio 9.
Dado un (1) número binario de cuatro (4) dígitos imprimir su bit da paridad. El bit de paridad es 1 si el número de bits 1 es impar y 0 en caso contrario.

Ejercicio 10.
Dado un (1) número binario de cuatro (4) dígitos imprimir su valor

Ejercicio 11.
Dado un (1) número de cuatro (4) dígitos imprimirlo por separado en unidades, decenas, centenas y unidades de mil.

Entrada:
1234

Salida:
Unidades: 4
Decenas: 3
Centenas: 2
Unidades de mil: 1

Estructuras De Control de Flujo de Datos

Ejercicio 1.

Todos los años que se dividen exactamente entre 400, o que son divisibles exactamente entre 4 y no son divisibles exactamente entre 100 son años bisiestos. Usando estas premisas crea un algoritmo que lea una fecha como un número entero con el formato **ddmmaaaa**, y luego extraiga el año de la fecha indicando si el mismo es un año bisiesto o no.

Ejercicio 2.

Dado un número entero cuya cantidad de dígitos es igual a 5, determine si es capicúa.
Nota: un número capicúa es aquel que se lee igual hacia adelante que hacia atrás

Ejercicio 3.

Cree un algoritmo que tome por entrada las horas y minutos de un día y dé como resultado su equivalente en segundos.

Ejercicio 4.

Para un valor entero positivo que representa una cantidad en segundos, indicar su equivalente en minutos, horas y días.

Ejercicio 5.

Dados tres números enteros positivos A, B y C, determine ¿cuál de ellos es el mayor? y ¿cuál es el segundo mayor?

Ejercicio 6.

En un estacionamiento el monto a pagar se calcula multiplicando el número de horas que permaneció el automóvil dentro del estacionamiento por Bs. 4 y se incrementa esta cantidad en Bs. 2,50 por cada media hora adicional.
Ahora se desea que usted elabore un algoritmo que a partir de la hora de entrada y la hora de salida de un vehículo (las mismas corresponden a un mismo día) calcule el monto a pagar por el dueño del vehículo.
La entrada vendrá dada por dos enteros positivos el primero representa las horas y el segundo los minutos, además por último se debe leer un carácter (A o P) que indica si la hora es AM o PM.

Ejercicio 7.

El IMC resulta de la división de la masa del individuo (en kilogramos) entre el cuadrado de la estatura (en metros). El índice de masa corporal es un indicador del peso de una persona en relación con su altura.
Clasificación del IMC de acuerdo con la OMS de la ONU:
a. Menor a 16: Criterio de ingreso.
b. 16 a 16.9: infra peso.
c. 17 a 18.4: bajo peso.
d. 18.5 a 24.9: peso normal.
e. 25 a 29.9: sobrepeso.
f. 30 a 34.9: obesidad pre-mórbida.
g. 40 a 45: obesidad mórbida.
h. Mayor a 45: obesidad híper-mórbida.

Dado el peso de una persona en libras (1 libra = 0,453592 Kg) y su estatura en centímetros, calcule su IMC e indique como salida el peso en kilogramos, el valor de IMC de la persona y la categoría en la cual fue clasificado.

Ejercicio 8.

Escriba un algoritmo que reciba una fecha (día y mes) correspondiente al año 2014 e imprima por pantalla el número de días que han pasado desde el 1 de Enero de 2014 hasta la fecha dada.

Ejercicio 9.

Solicitar un número entre el 1 y el 12 e imprimir el mes correspondiente a dicho número.

Ejercicio 10.
En un almacén se hace un 20% de descuento a los clientes cuya compra supere los Bs 1000, se desea que realice un algoritmo el cual tome por entrada el monto a pagar por el cliente y arroje como salida el monto aplicando el descuento de ser necesario.

Estructuras Iterativas

Ejercicio 1.

Dado un número entero N, calcular e informar al usuario cuántos dígitos tiene dicho número.

Ejercicio 2.

Dado un número, determine si es capicúa.
Nota: un número capicúa es aquel que se lee igual hacia adelante que hacia atrás.

Ejercicio 3.

Dado un número N determinar si es un número primo.
Nota: Un número primo es aquel que solo es divisible por 1(uno) y por el mismo.

Ejercicio 4.

Construya un programa que dado un valor entero N, haga el cálculo de la función factorial utilizando estructuras iterativas.

Ejercicio 5.

Dado un número entero N que representa una contraseña y asumiendo que una contraseña debe tener al menos 10 dígitos para ser segura, determine si la contraseña ingresada por el usuario es correcta, de no serlo debe pedirla nuevamente hasta que tenga los 10 (diez) dígitos solicitados y al ser correcta mostrar un mensaje de éxito al usuario, por salida estándar.

Ejercicio 6.

Dada una secuencia de números terminada en cero (0), elaborar un algoritmo que informe al usuario qué valor tiene el número mayor y qué valor tiene el número menor, sin contar el cero (0).

Ejercicio 7.
Se tiene una secuencia de enteros terminada en cero, que corresponde a la edad, peso y estatura de una muestra de hombres y mujeres mayores de 18 años. Con base en dicha secuencia se desea realizar un estudio a fin de conocer:

- Edad promedio de todas las personas en la muestra.
- Peso promedio de todas las personas en la muestra.
- Estatura promedio de todas las personas en la muestra.
- Cuántas personas hay con edad entre los 18 y 25 años.
- Cuántas personas son mayores a 36 años.

- Cuál es el promedio de peso de las personas con edades entre 18 y 35 años.

Ejercicio 8.

Construye un algoritmo que calcule e imprima la tabla de multiplicar, desde la tabla del 1 hasta la del 10.

Ejercicio 9.
Escribir un algoritmo que muestre todas las fichas de dominó, sin repetir.

Ejercicio 10.

Dados N número positivos (N>1) calcular el promedio de esta serie. Considere que la serie termina al leer un 0.

Procedimientos (Acciones y Funciones)

Ejercicio 1.

Construya una función que solicite edades al usuario y determine el promedio de las edades mayores a 18 años. El usuario indicará cuando desea dejar de suministrar datos de entrada. En la Acción Principal se informará el promedio calculado.

Ejercicio 2.

Construya una función "Eleva" Que reciba como parámetros una base y un exponente y retorne al algoritmo principal el resultado de elevar un numero al otro.

Ejercicio 3.

Escriba una función que calcule el perímetro del pentágono (siendo el perímetro la suma de los lados del polígono).

Ejercicio 4.

En una empresa pagan según las horas trabajadas y una tarifa fija por hora. Si la cantidad de horas trabajadas en una semana es mayor a 40, la tarifa se incrementa en un 35% para las horas extras. Escribe una acción principal que solicite la identificación de 5 empleados, el monto cancelado por hora, y la cantidad de horas trabajadas por cada uno, llame a acciones/funciones que calculen el salario semanal por horas trabajadas (<=40) y los ingresos por concepto de horas extras, y finalmente informe los resultados en la acción principal.

Ejercicio 5.

Escriba una acción (MillasAKilometros) que convierta una distancia en millas a kilómetros (1milla = 1.60935km). Esta acción recibirá como parámetros: el nombre de la ciudad origen, el nombre de la ciudad destino y la distancia en millas entre ellas y debe retornar la distancia entre las ciudades en kilómetros.
Desarrolle además una acción principal en donde utilice a la acción MillasAKilometros para convertir e informar la distancia en kilómetros entre 4 pares de ciudades suministradas por el usuario.

Entrada:

Cuidad A
Ciudad B
332

Salida:

Entre la Ciudad A y la Ciudad B hay 534.30 Km.