**COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 2 |
| NOMBRE DEL COMPONENTE FORMATIVO | Análisis y solución de problemas aplicando algoritmos |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | Implementar algoritmos en la resolución de procesos lógicos utilizando lenguaje de programación seleccionado. |

1. **Metodologías de algoritmos**

De manera implícita, en las distintas áreas productivas de la sociedad se usan algoritmos y un par de ejemplos pueden ser que, en la construcción de una casa, el ingeniero o el arquitecto analiza el terreno, diseña unos planos, realiza maquetas y así continúan siguiendo una secuencia lógica de procesos, para la resolución del problema; o un abogado que antes de defender a un cliente analiza las pruebas, elabora un plan, practica sus argumentos y sigue un método para lograr la defensa de su cliente. Se puede concluir, entonces, que una solución a la medida de un problema requiere una metodología que lleve a la solución final.

En el proceso de desarrollo de programas informáticos, aparte de conocer esta metodología, se necesita conocer las funciones que puede realizar un computador y la representación de las mismas.

En general, existen tres (3) fases secuenciales para la elaboración de un algoritmo representadas en la siguiente figura y de las cuales se abordarán las dos primeras:

**Figura 1**  
*Metodología de creación de algoritmos*

1. **Análisis**

Como se percibió en componentes anteriores, la fase de análisis trata de responder a la pregunta: ¿qué es lo que se debe hacer?, y esto es necesario entender el problema. Por ello, es preciso relacionar los datos de entrada, salida o resultados con los procesos a ejecutar con esta información y los datos auxiliares que se puedan requerir en las tareas.

Hasta este punto parece que esas son las actividades propias del análisis, sin embargo, el éxito de un algoritmo consiste en reducir a cero la brecha de lo que el cliente necesita y los resultados que el algoritmo entrega. Para lograrlo en su totalidad, es importante indagar otros aspectos que corresponden a la naturaleza del problema, las siguientes preguntas modeladoras son de referencia casi obligante a tener en cuenta:

|  |
| --- |
| **¿Qué es? ¿Quiénes intervienen? ¿Cómo interviene o se comporta? ¿Cuándo? ¿Cómo lo hace? ¿Por qué? y ¿Para qué?** |

1. **Diseño**

En esta etapa se responde, básicamente, a la pregunta **¿cómo se va a hacer?**, refiriéndose a la manera en la que el sistema va a realizar la tarea solicitada. Lo importante en este punto, es que debe quedar bien definido el problema, sin dar espacio a más de una interpretación de la acción o acciones que debe hacer el sistema.

En el proceso de desarrollo del algoritmo, las siguientes actividades son obligatorias, sobre todo si se está empezado en el diseño de soluciones basadas en algoritmos y no se cuenta con mucha experiencia:

* Definir las partes del algoritmo.
* Pasos para el desarrollo del algoritmo.
* La representación del algoritmo (notación, diagrama o seudocódigo).

Para la verificación del algoritmo,se aplica una técnica denominada pruebas de escritorio o trazas, que trata de la validación y verificación del algoritmo mediante la ejecución de las sentencias u operaciones que lo componen (proceso), y así determinar sus resultados de salida a partir de una base determinada de elementos en la entrada del proceso.

Como el resultado del proceso de diseño es un algoritmo, es obligatorio saber representarlo de alguna manera, y se empleará inicialmente con la notación de seudocódigo para hacer este tipo de representaciones.

1. **Notación de diseño: seudocódigo**

Para expresar el contenido de un programa de computadora, se necesita de un lenguaje de programación que las computadoras pueden interpretar. Sin embargo, se puede expresar ese mismo contenido en el lenguaje natural lengua materna (español), empleando expresiones sencillas que lo simplifican y abrevian. Cuando se emplea esta técnica, se dice que está escrito en seudocódigo.

Ejemplo: el cliente pide que se desarrolle un programa de computadora que le solicite al usuario un número, luego el sistema debe mostrar en pantalla el número digitado.

A continuación, se aplica la metodología vista:

* **Análisis:** el dato de entrada debe ser un número ingresado por el usuario, pero antes de eso el sistema debe solicitarle al usuario que lo ingrese (digite) al programa, si no, ¿de qué otra manera el usuario sabría que el programa está esperando el ingreso de ese dato? Por lo tanto, una posible operación inicial dentro del algoritmo sería escribir en pantalla que “*por favor se ingrese un número”*.

La segunda operación es la de recibir ese número desde el teclado, para ingresarlo al programa, pero esto implica que se debe definir un dato intermedio donde se guardará ese valor y el cual identificará el valor ingresado (a este dato se denomina variable porque puede tener cualquier valor entero según digite el usuario). Ya ocurre el primer problema: para guardar el dato (variable) se debe definir dónde se almacenará o como podría referirse al dato ingresado dentro del programa, por lo tanto, antes de empezar el algoritmo se debe indicar al computador que reserve un espacio para guardar un número entero.

Una vez definida la variable, es posible referirse a ella, leer desde el teclado el número entero y almacenarlo en dicha variable.

Por último, el sistema debe imprimir el contenido de la variable (dato de salida), con lo cual se puede dar fin al programa.

* **Diseño:** en anterior análisis se contemplaron los datos de entrada, datos auxiliares (variable) operaciones (impresión en pantalla, lectura de datos e impresión en pantalla de resultados). Pero son muchas las palabras usadas en el análisis por lo tanto lo abreviamos con el siguiente código que, posteriormente, se explicará.

**Figura 2**  
*Seudocódigo imprimir un número*

**ALGORITMO** Mostrar número digitado;

**VAR**

**ENTERO** entrada;

**INICIO**

**ESCRIBIR** ("Por favor ingrese un número");

**LEER** (entrada);

**ESCRIBIR** (entrada);

**FIN**

En **ALGORITMO** se indica el nombre del algoritmo, y después se declaran las variables que serán usadas en él (con **VAR**) indicando su tipo de dato (un valor de tipo **ENTERO**, para este ejemplo).

Entre **INICIO** y **FIN** se escriben las diferentes acciones que realizará el algoritmo y usamos **ESCRIBIR**() para indicar con esta palabra que se mostrará un mensaje por pantalla (el texto que se va a mostrar 'tal cual' debe ir entre paréntesis dobles), y **LEER**() para recibir el dato desde el teclado.

Se debe tener en cuenta que cada línea termina en punto y coma (excepto **INICIO** y **FIN**), el punto y coma indica que lo que se ejecuta es una sentencia que hace una única operación, mientras que lo que está entre **INICIO** y **FIN** agrupa varias sentencias que se ejecutan en un orden determinado.

1. **Pruebas de escritorio o trazas**

Las pruebas de escritorio son las simulaciones de ejecución de un algoritmo que permite determinar la validez o efectividad del mismo. Se trata de escribir en una tabla con tantas columnas como variables tiene el algoritmo y seguir las instrucciones poniendo los valores correspondientes.

Ejemplo: escribir un algoritmo que: calcule el cuadrado y el cubo de un número introducido por teclado y mostrar los resultados para el cubo y el cuadrado.

* **Análisis: nuevamente** el dato de entrada debe ser un número ingresado por el usuario, y al igual que en el ejemplo anterior se debe solicitar al usuario que lo digite.

Antes de realizar las operaciones de elevar al cuadrado y al cubo, se necesita definir las dos variables que recibirán estos datos (el cuadrado, y el cubo), por lo tanto, el algoritmo debe definirlas antes de realizar las operaciones y asignarles a ellas los resultados.

Una vez realizadas las operaciones el algoritmo debe mostrar en pantalla el contenido de estas variables, con un mensaje texto que indique si es el resultado del cuadrado o del cubo.

* **Diseño:** la siguiente figura tiene la representación del resultado al análisis anterior.

**Figura 3**  
*Metodología de creación de algoritmos*

**ALGORITMO** Calcular el cuadrado y el cubo;

**VAR**

**ENTERO** entrada;

**ENTERO** cuadrado;

**ENTERO** cubo;

**INICIO**

**ESCRIBIR**("Por favor ingrese un número");

**LEER**( entrada );

    cuadrado <= entrada \* entrada;

    cubo <= entrada \* entrada \* entrada;

**ESCRIBIR**( "El Cuadrado " cuadrado );

**ESCRIBIR**( "El Cubo " cubo );

**FIN**

Las pruebas de escritorio permiten verificar que el pseudocódigo realice lo que se espera al resolver el problema. Sin embargo, por ahora es importante que a todos los pseudocódigos se les aplique, al menos, una de prueba de escritorio o las que se consideren necesarias.

La mayoría de las pruebas de escritorio se representan en una tabla, donde en cada columna se anotan las variables y condiciones si las hay, en el orden en que aparecen en el algoritmo, para entenderlo se tiene que recorrer línea a línea el pseudocódigo y así identificar tanto las variables como condiciones e ir anotándolas en la tabla de la prueba de escritorio.

Es importante que en la tabla no aparezcan variables o condiciones repetidas y en la última columna se pueden anotar los datos de salida para entonces también anotar “Salida”.

Ejecutando línea a línea el algoritmo de la figura 3 y se va llenando la tabla como muestra la siguiente figura:

**Figura 4**  
*Tablas de traza del algoritmo*

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

También es posible realizar pruebas de escritorio a otras representaciones de algoritmos, como los diagramas de flujo, expuestos a continuación, y que se representan con algoritmos recomendando que a cada ejemplo presentado se le realice la prueba de escritorio correspondiente.

1. **Diagramas de flujo**

Una forma común de presentar los algoritmos es con diagramas de flujo, que consiste en una representación por medio de un gráfico. Mientras en seudocódigo las sentencias se representaban con preposiciones en diagramas de flujo la preposición es reemplazada por una figura.

La representación del algoritmo de la figura 4 sería:

**Figura 5**  
*Diagrama de flujo imprimir un entero*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **Inicio del algoritmo.**  **Muestra en pantalla el mensaje que ingrese un número.**  **Leemos del teclado un número.**  **Muestra en pantalla el valor del número digitado.**    **Final del algoritmo.** |

La representación del algoritmo de la figura 3 (algoritmo que calcule el cuadrado y el cubo de un número introducido por teclado y mostrar los resultados el cubo y el cuadrado) es:

**Figura 6**  
*Diagrama de flujo imprimir el cuadrado y el cubo de un entero*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **Inicio del algoritmo**  **Muestra en pantalla el mensaje que ingrese un número**    **Leemos del teclado un número**  **Calcula el cuadrado del número digitado**  **Calcula el cubo del número digitado**    **Muestra el cuadrado del número digitado**    **Muestra el cubo del número digitado**  **Final del algoritmo** |

La Organización Internacional para la Estandarización (ISO, por su sigla en inglés) y el Instituto Nacional Americano de Estandarización (ANSI, por su sigla en inglés), estandarizaron los símbolos que han sido mayor mente aceptados en 1985. Con el fin de evitar la utilización de símbolos diferentes para representar procesos iguales (López, 2009).

Los siguientes son los principales símbolos para crear diagramas de flujo.

**Tabla 1**  
*Símbolos de los diagramas de flujo*

**Diagrama

Descripción generada automáticamente**

1. **Herramientas digitales para creación y prueba de algoritmos**

Para aprender a reconocer los elementos que componen los diagramas de flujo se puede hacer uso de la herramienta denominada Draw.io a través del enlace <https://app.diagrams.net/> con la que puede usar una cuenta de correo de Google, o misena.edu.co para acceder y crear un nuevo diagrama.

Es ideal ingresar y seleccionar los diagramas que se muestran en la siguiente figura.

**Figura 7**  
*Seleccionar formas*

|  |  |
| --- | --- |
| **Texto  Descripción generada automáticamente con confianza media** | **Diagrama, Forma  Descripción generada automáticamente** |

Nota. Tomada de <https://app.diagrams.net/>

En caso de contar con intermitencia de internet se puede realizar el ejercicio propuesto a través del programa de libre distribución disponible en <http://dia-installer.de/> o puede acceder a la versión de escritorio de Draw.io disponible en <https://app.diagrams.net/>.

|  |
| --- |
| **Instituto de Informática Uach. (08 de septiembre de 2020). *Cómo descargar draw io 2 para dibujar diagramas de flujo.* [Video]. YouTube.** [**https://www.youtube.com/watch?v=encz3h8TLWA**](https://www.youtube.com/watch?v=encz3h8TLWA) |

El programa de mayor uso para el diseño y depuración de algoritmos en seudocódigo es PSeInt disponible en <http://pseint.sourceforge.net/>.

1. **Uso de identificadores y palabras reservadas**

En la elaboración de diagramas de flujo o algoritmos muchas veces se necesitan datos auxiliares, a continuación, se definen algunos de estos tipos de datos auxiliares.

Los identificadores son nombres que se dan a los elementos utilizados para resolver un problema y poder diferenciar unos de otros. De acuerdo con López (2009), al asignar nombres (identificadores) a variables, constantes y procedimientos se deben tener en cuenta algunas reglas, a saber:

* Pueden estar formados por una combinación de letras y números (saldoMes, salario, fecha2, etc.).
* El primer carácter de un nombre debe ser una letra.
* La mayoría de los lenguajes de programación diferencian las mayúsculas de las minúsculas.
* Los nombres deben ser nemotécnicos, con solo leerlos se puede entender lo que contienen.
* Deben ser muy descriptivos; no utilizar abreviaturas, a menos que se justifique.
* Es conveniente utilizar una sola palabra para nombrar páginas, controles, variables, etc.
* No utilizar caracteres reservados (%, +, /, >, etc.) tampoco letras acentuadas (á, é, í, ó, ú).
* No utilizar palabras reservadas por los lenguajes de programación.

|  |
| --- |
| **Ejemplo: listar los identificadores usados en los algoritmos previos.** |

**Solución**: en el algoritmo de imprimir un número entero usando el identificador de **entrada** para representar el número digitado.

En el algoritmo de imprimir el cuadrado y el cubo de un número entero empleando el identificador de **entrada** para representar el número digitado, el identificador cuadrado para representar el valor del **cuadrado** de entrada y el identificador **cubo** para almacenar el cubo de **entrada**.

Todos los lenguajes de programación tienen unas palabras para nombrar sus comandos, instrucciones y funciones; estas palabras no se pueden usar como indicadores.

1. **Variables**

Para los algoritmos con diversos conjuntos de datos iniciales, se debe determinar una independencia entre los datos iniciales de un problema y la estructura de su solución. Esto se hace mediante la utilización de variables (cantidades que se suelen emplear y que durante la ejecución del algoritmo pueden tomar cualquier valor de un intervalo de valores posibles).

Las variables son espacios de trabajo (contenedores) dispuestos para almacenar datos (valores). El valor de una variable puede ir cambiando en algún paso del algoritmo o permanecer sin cambios; por lo tanto, el valor de una variable es el del último valor o dato asignado a esta. En la figura 6, las variables eran los indicadores **cuadrados** y **cubo**.

1. **Constantes**

Las constantes se crean de la misma forma que las variables, a diferencia que se procura su definición o creación a inicio del algoritmo. Las constantes consisten en datos que, luego de ser asignados, no cambian en ninguna instrucción del algoritmo; pueden contener constantes matemáticas (pi).

1. **Contadores**

Un contador es una variable que está en ambos lados de una asignación (a la derecha y a la izquierda) interna, a la que se le suma un valor constante. Es necesario haberla iniciado al principio del algoritmo, ya que va a ser leído su contenido.

|  |
| --- |
| **Ejemplo: definir un contador llamado contador que se incremente en uno cada vez tres veces.** |

**Figura 8**  
*Contador en un algoritmo*

**ALGORITMO** Contador;

**VAR**

**ENTERO** contador;

**INICIO**

contador= 0;

    contador <= contador + 1;

    contador <= contador + 1;

    contador <= contador + 1;

**ESCRIBIR**( contador );

**FIN**

Como se puede ver en el algoritmo, tres veces se incremente en 1 (uno) el valor del de contador y al final de la ejecución su valor es de 3. Es importante notar que contador inicia con un valor el valor 0, si esta asignación inicial no se hace el computador pondría un valor aleatorio y el valor al final del algoritmo sería indeterminado.

1. **Acumuladores**

Los acumuladores son una versión ampliada de contadores, tienen las mismas características que un contador excepto el valor que se va incrementando no es una constante sino una que es un valor variable.

1. **Operadores y jerarquía en los operadores**

Los operadores son símbolos que sirven para manipular datos. Y en un algoritmo es necesario dejar un espacio en blanco a cada lado del signo aritmético. Los operadores y las funciones que realizan se clasifican como se indica a continuación:

* **Aritméticos**: permiten realizar operaciones entre datos de tipo numérico y dan como resultado otro valor de tipo numérico. Ejemplo: producto o multiplicación (\*); división (/); suma (+); resta (-); asignación (=).
* **Alfanuméricos**: permiten operar con datos de tipo carácter (letras) o cadenas de texto. La mayoría de los lenguajes de programación admiten el operador + para realizar la concatenación (unión) de caracteres o cadenas.
* **Relacionales**: permiten la comparación entre datos del mismo tipo de dato y dan como resultado dos valores posibles: verdadero o falso. Ejemplo: igual a (=); menor que (<); mayor que (>,) mayor e igual que (>=).
* **Lógicos**: posibilitan la evaluación lógica de dos expresiones de tipo lógico. Dan como resultado uno de dos valores posibles: verdadero o falso. Ejemplo: negación (no); conjunción (y); disyunción(o).

1. **Jerarquía en los operadores**

Las computadoras ejecutan los operadores en un orden establecido y el siguiente es el orden (jerarquía) para ejecutar operadores:

1. Paréntesis (se ejecutan primero los más internos).

2. Signo (-2), si un valor es positivo o negativo.

3. Potencias (^) y Raíces (sqrt); Productos y Divisiones (\* y /), en este mismo orden.

4. Sumas y Restas (+ y -).

5. Concatenación (+).

6. Relaciónales (=, <, >).

7. Negación (no).

8. Conjunción (y).

9. Disyunción (o).

El siguiente ejemplo permite comprender mejor la jerarquía de operadores:

**Tabla 2**  
*Orden de ejecución de una expresión matemática*

|  |  |  |
| --- | --- | --- |
| **Orden** | **Operación** | **Explicación y expresión resultante** |
| 1 | **7 + 8** | Primero que todo lo que más valor tiene en la jerarquía son los paréntesis de manera que y en particular los más interiores de esta forma el computador se interesara en resolver primero el contenido de (**7 + 8).** Y el resultado será **15.**  De esta forma, la expresión queda reducida a**:**  **(2 + 15 \* 16 + 3^2) /4 +5** |
| 2 | **3^2** | De la expresión resultante luego de la operación 1, se observa que nuevamente hay un paréntesis por reducir, de manera que la computadora intentará resolver su contenido primero, como dentro del paréntesis está la expresión de elevar al cuadrado el número 3 (**3^2**) que tiene como resultado 9, la expresión resultante sería:  **(2 + 15 \* 16 + 9) /4 +5** |
| 3 | **15 \* 16** | De los paréntesis de la expresión resultante la operación de más jerarquía es la multiplicación **15 \* 16** que tiene como resultado 240, la expresión resultante sería:  **(2 + 240 + 9) /4 +5** |
| 4 | **2+24+9** | De la expresión resultante el que tiene mayor precedencia es el paréntesis por lo tanto la suma aritmética de su contenido data como expresión resultante:  **251/4 +5** |
| 5 | **251/4** | De la expresión resultante el operador de mayor jerarquía es la división y da como resultado 62.75 que en la siguiente operación se le sumará el valor de 5 la expresión resultante será:  **62,75 +5** |
| 6 | **62,75 +5** | **67,75** |

A continuación, se revisan dos ejemplos:

**Ejemplo 1:**

|  |
| --- |
| **Una computadora debe ejecutar las siguientes operaciones, se debe escribir en el orden que la computadora ejecutará las operaciones de la siguiente expresión matemática: (2 + (7 + 8) \* 16 + 3^2) /4 +5.**  **Solución: primero que todo, lo que más valor tiene en la jerarquía son los paréntesis de manera que y en particular los más interiores, de esta forma la computadora se interesara en resolver primero el contenido de (7 + 8).** |

**Ejemplo 2**:

|  |
| --- |
| **Escribir la siguiente ecuación como una expresión que la computadora pueda interpretar en el orden adecuado.  Diagrama  Descripción generada automáticamente con confianza baja**  **Solución:**  **sqrt (6^2/2 + 7) / 8^2** |

1. **Estructuras de control secuencial**

Una estructura es definida como un esquema con determinada distribución y orden que permite representar una idea de forma simplificada. La estructura de control secuencial es la más sencilla y es conocida como estructura lineal. Se compone de instrucciones que deben ejecutarse en forma consecutiva, una tras otra, siguiendo una línea de flujo. Los problemas más sencillos pueden resolverse haciendo uso únicamente de esta estructura. Por lo general, la estructura secuencial forma parte de soluciones a problemas complejos, en las que se la utilizan otras estructuras iterativas (repetir varias veces un conjunto de instrucciones) y condicionales (López, 2009).

La siguiente figura muestra la estructura general del algoritmo más básico.

**Figura 9**  
*Modelo de estructura secuencial*

Diagrama

Descripción generada automáticamente

1. **Ejemplo estructura de control secuencial**

Escribir un algoritmo para calcular el área de cualquier triángulo rectángulo, en él se debe pedir al usuario que ingrese los valores de la altura y la base del triángulo

**Análisis:**

* **Formular el problema**: está claramente planteado y consiste en calcular el área de un rectángulo.
* **Resultados esperados**: se espera que el algoritmo sea capaz de calcular el área del rectángulo y mostrar en la pantalla el resultado de ese cálculo.
* **Datos disponibles**: dos datos números que son la base y la altura del triángulo (se deben solicitar al usuario). El aprendiz debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema, de no ser así, tendrá plantear una estrategia para obtener los conocimientos requeridos. Para este ejemplo recordar que un triángulo rectángulo es aquel que uno de sus lados tiene un ángulo recto (90 grados).
* **Restricciones**: los valores de base y altura son variables y se deben solicitar al usuario.
* **Procesos necesarios**: definir variables; asignar el valor 2 a la constante usando para dividir (***divisor***); solicitar al usuario el valor de la altura del triángulo; solicitar al usuario el valor de la base; aplicar la fórmula de área; mostrar el resultado.

**Diseño**

El algoritmo resultado del análisis se puede ver en la figura 8, en la que están representados tanto el diagrama de flujo como en seudocódigo, mientras se está aprendiendo a diseñarlos, es importante hacerlos algoritmos en diagrama de flujo y en la medida que se va adquiriendo experticia con la representación de seudocódigo es más conveniente hacer esta representación en razón de la velocidad a la hora de diseñarlo.

**Figura 10**  
*Diagrama al área de un triángulo*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente**  *.* | **ALGORITMO** área de un triángulo**;**  **VAR**  **ENTERO** divisor**;**  **REAL** area**;**  **REAL** base**;**  **REAL** altura**;**  **INICIO**  divisor = 2;  **ESCRIBIR(** "Escriba la base "**);**  **LEER(** base **)**  **ESCRIBIR(** "Escriba la altura "**);**  **LEER(** altura **);**   area = base \* altura / divisor;  **ESCRIBIR(** area **);**  **FIN** |
|  |

Del diagrama de flujo y el seudocódigo se puede notar lo siguiente: el identificador de áreano tiene tilde por la razón antes mencionada, el tipo de dato REALhacereferenciaaque es un número que no solamente puede contener un número entero, sino también un valor con cifras decimales.

1. **Estructura condicional**

Es importante prestar atención a las estructuras que utilizan para resolver problemas y poderlas reconocer para lograr mayor control sobre la solución planteada.

La estructura condicional se emplea para indicarle al computador que debe evaluar una condición y, según el resultado, ejecutar el bloque de instrucciones. La forma más común está compuesta por una condición que se evalúa y dos bloques de instrucciones que se ejecutan: uno cuando la condición es verdadera y otro cuando la condición es falsa.

|  |
| --- |
| **Ejemplo: un aprendiz aprueba un examen cuando la calificación de este es mayor o igual a 3. Elaborar un algoritmo que pida al usuario una calificación, aplicar el criterio de aprobación e imprimir “Aprobado” o “Reprobado”, según sea el caso.** |

**Análisis**

**Formular el problema**: es un problema sencillo de condición con dos operaciones o condición doble.

**Resultados esperados**: un aviso que reporte si el estudiante obtiene “Aprobado” o “Reprobado” el examen.

**Datos disponibles**: la calificación ingresada por el usuario. Para aprobar, la nota debe ser mayor o igual a 3.

**Restricciones**: aplicar el criterio de aprobación (calificación mayor o igual a 3).

**Procesos necesarios**: solicitar al usuario que ingrese la calificación. Evaluar si la calificación es igual o superior a 3; en caso de ser verdadero, reportar “Aprobado”; en caso contrario, reportar “Reprobado”.

**Diseño**

**Figura 11**  
*Estructura condicional doble*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO** Calificación del usuario;  **VAR**  **REAL** calificacion  **INICIO**  **ESCRIBIR(** "Escriba la calificación"**);**  **LEER(** calificacion **)**  **SI (** calificación >= 3**)**  **ESCRIBIR(** "Aprobado" **);**  **SI NO**  **ESCRIBIR(** "Reprobado" **);**  **FINSI**  **FIN** |
|  |

En la figura 11 se observa cómo se introduce la calificación por teclado, tanto en el diagrama de flujo (con la representación de trapecio) o el seudocódigo con la preposición **LEER**, luego este valor se compara en un condicional, en el diagrama de flujo con la representación de un rombo, mientras que en el seudocódigo se una la preposición **SI**, seguido de las preposiciones que se han de ejecutar en caso de que la condición sea verdadera (note que está escrita más a la derecha). También se usa la preposición **SI NO** para identificar el inicio de bloque de instrucciones a ejecutar si no se cumple la condición y la preposición **FINSI** para indicar que no hay más intrusiones que ejecutar si se cumple o no la condición.

Como la condición evalúa dos condiciones si se cumple o no, y en cada caso existe un bloque de instrucciones (o alimenta una instrucción) a seguir se llama: ESTRUCTURA CONDICIONAL SIMPLE, la figura 9 corresponde a una de estas condicionales dobles, ya que si la condición se cumple escribe algo y si no se cumple también hace otra operación.

|  |
| --- |
| **Ejemplo: escribir un algoritmo en donde se pida la edad del usuario; si es mayor de edad debe aparecer un mensaje indicándolo.** |

**Análisis**

**Formular el problema**: es un problema sencillo de condición simple.

**Resultados esperados**: un mensaje que indique si el usuario es mayor de edad si no es mayor de edad no hacer nada.

**Datos disponibles**: la edad del usuario en años (número entero).

**Restricciones**: aplicar el criterio de mayoría de edad (edad mayor o igual a 18).

**Procesos necesarios**: solicitar al usuario que ingrese su edad. Evaluar si la edad es igual o superior a 18; en caso de ser verdadero “usted es mayor de edad”.

**Diseño**

**Figura 12***Estructura condicional simple*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO** Edad del usuario;  **VAR**  **REAL** edad;  **INICIO**  **ESCRIBIR(** "Escriba la calificación"**);**  **LEER(** edad **)**  **SI (** edad >= 18**)**  **ESCRIBIR(** "Usted es mayor de edad" **);**  **FINSI**  **FIN** |
|  |

Como observa en la figura 12, no se ejecuta nada en caso de que NO se cumpla la condición. Este tipo de estructura condicional se denomina ESTRUCTURA CONDICIONAL SIMPLE; además, todo el proceso (pedir la edad, comprobar la condición, mostrar o no el mensaje) se realiza una única vez y finaliza la ejecución del algoritmo, si se deseara que el programa repitiera estas mismas operaciones más de una vez, se tendría que usar una estructura de iteración como se verá a continuación.

1. **Estructuras de iteración o repetitivas**

Las estructuras iterativas o de repetición permiten ejecutar una o varias instrucciones, un número determinado de veces o indefinidamente hasta que se cumpla una condición. Esta estructura ayuda a simplificar los algoritmos.

En programación existen, como mínimo, dos tipos de estructuras repetitivas las cuales tienen variantes en los diferentes lenguajes de programación.

La característica común es que ambos tipos permiten ejecutar una o varias instrucciones:

* Un número conocido de veces (Estructura PARA o FOR en inglés).
* Mientras se cumpla una condición (Estructura MIENTRAS o WHILE).

1. **Estructura PARA (FOR)**

Esta estructura se usa cuando se sabe el número de veces que se debe ejecutar un conjunto de operaciones.

|  |
| --- |
| **EJEMPLO: escribir un procedimiento que muestre siete (7) veces en pantalla la frase “Esto es un algoritmo”. Mostrar en pantalla la salida del en el computador.** |

**Análisis**

**Formular el problema**: es un problema sencillo solo consiste en usar una sentencia de repetición.

**Resultados esperados**: el mensaje “Esto es un algoritmo” 7 veces impreso en pantalla.

**Datos disponibles**: constante con el número de repeticiones (7).

**Restricciones**: ninguna.

**Procesos necesarios**: hacer que se imprima en pantalla el mensaje 7 veces

**Diseño**

**Figura 13**  
*Estructura de repetición FOR*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO** Ciclo para;  **INICIO**  **PARA**  i **DESDE** 1 **HASTA** 7  **ESCRIBIR(** "Esto es un algoritmo" **);**  **FINPARA**  **FIN** |
|  |

La salida en pantalla sería:

Esto es un algoritmo

Esto es un algoritmo

Esto es un algoritmo

Esto es un algoritmo

Esto es un algoritmo

Esto es un algoritmo

Esto es un algoritmo

Como se puede ver en el algoritmo de la figura 13, existe una variable **i** tendrá el valor 1, 2, 3 ... hasta 7, esta variable, se puede usar dentro del el bloque **PARA** hasta **FINPARA**.

Ahora se presenta un ejemplo un poco más elaborado.

|  |
| --- |
| **Ejemplo: se necesita elaborar un algoritmo que solicite al usuario un número entero y sume todos los números naturales que hay hasta ese número. Por ejemplo, si el usuario digita 3, el programa debe sumar: 1 + 2 + 3, si el usuario digita 5 el programa debe sumar 1+2+3+4+5. Al finalizar debe imprimir el resultado.** |

**Análisis**

**Formular el problema**: necesita la sumatoria de los números en secuencia natural.

**Resultados esperados**: la suma de los números en sucesivos de uno hasta el número digitado por el usuario.

**Datos disponibles**: el dato entero que digita el usuario es la entrada.

**Datos adicionales**: se necesita un acumulador donde ir guardando el valor de las sumas sucesivas.

**Restricciones**: ninguna.

**Procesos necesarios**: definir un acumulador con valor inicial cero, imprimir para que el usuario sepa que debe digitar un número (“Escriba el número”), Leer del teclado el número del usuario, sumar cada vez el dato que se va incrementando de uno en uno al valor digitado por el usuario.

**Diseño**

**Figura 14**  
*Estructura de repetición FOR acumulador suma*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO** sumas sucesivas;  **VAR**  **ENTERO** número**;**  **ENTERO** suma**;**  **INICIO**  **suma = 0;**  **ESCRIBIR(** "Escriba el número" **);**  **LEER(** número  **PARA** i **DESDE** 1 **HASTA** número  suma = suma + i;  **FINPARA**  **ESCRIBIR(** suma **);**  **FIN** |
|  |

En la anterior figura se muestra el algoritmo en sus dos representaciones más comunes, se propone en secciones posteriores hacer una prueba de escritorio a este problema.

1. **Estructura MIENTRAS (WHILE)**

La instrucción "Mientras", puede impedir la ejecución de un conjunto de instrucciones, si la evaluación de una condición lógica es falsa. Esto significa que es repetitiva únicamente cuando la evaluación de la condición es verdadera.

|  |
| --- |
| **Ejemplo: realice un programa que permita obtener la factorial de un número dado por el usuario.** |

**Análisis**

**Formular el problema**: la factorial de un entero positivo n, se define como el producto de todos los números enteros positivos desde 1 hasta n.

Ejemplo: ¡Matemáticamente 5! = 5\*4\*3\*2\*1 o a la inversa: 5! = 1\*2\*3\*4\*5.

**Resultados esperados**: la factorial del número digitado por el usuario.

**Figura 15**  
*Proceso de factorial*

Imagen que contiene Gráfico

Descripción generada automáticamente

En la anterior figura se observa que es preciso ir incrementando el valor en 1 (valor constante - contador -) hasta que llegue al número 5 (datos suministrados por el usuario); es decir, 1\*2 = 2, este resultado se multiplica por 3 (2 \* 3 = 6), 6 se multiplica por 4 (6 \* 4 = 24), finalmente también 24 se multiplica por 5 y se obtiene la factorial (24 \* 5 = 120), de esta forma, los resultados de las multiplicaciones se van acumulando hasta obtener la factorial.

**Análisis**

**Datos disponibles**: el dato entero que digita el usuario es la entrada.

**Datos adicionales**: se requiere de un acumulador donde ir guardando el valor de las multiplicaciones sucesivas, también un contador para ir contando el número secuencial (como lo hace el número i en el ciclo PARA).

**Restricciones**: la repetición se detiene cuando.

**Procesos necesarios**: definir un acumulador con valor inicial cero, imprimir para que el usuario sepa que debe digitar un número (“Por favor el número”), leer del teclado el número del usuario, multiplicar cada vez el dato que se va incrementando de uno en uno al valor digitado por el usuario.

**Diseño**

**Figura 16**  
*Algoritmo factorial ciclo mientras*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO** factorial**;**  **VAR**  **ENTERO** contador**;**  **ENTERO** factorial**;**  **INICIO**  contador = 1;     factorial = 1;  **ESCRIBIR(** "Escriba el número" **);**  **LEER(** numero **);**  **MIENTRAS** contador <= numero **HACER**   factorial = factorial \* contador;         contador = contador + 1;  **FINMIENTRAS**  **ESCRIBIR(** factorial **);**  **FIN** |
|  |

*.*

Como se puede observar, si se hace la prueba de escritorio se verá cómo el proceso logra el objetivo, también se debe anotar que este algoritmo se puede hacer con una sentencia repetitiva PARA, de manera tal que queda como ejercicio.

Las estructuras de control repetitivas son muy útiles y necesarias a la hora de manipular tipos de datos más complejos que los hasta ahora vistos, como los arreglos (vectores) o las matrices, ya que estos tipos de datos requieren realizar operaciones que se repiten más de una vez.

1. **Estructura de datos**

Es importante conocer el concepto de arreglos e identificar cuándo usarlos en la aplicación en los algoritmos. Así mismo, ser capaz de resolver problemas básicos mediante diagramas de flujo y seudocódigo. De otra parte, existen varios tipos de arreglos y de estos los más comunes son los vectores y las matrices.

1. **Vectores**

Los arreglos o *array*, en inglés, (matriz, arreglo o vector) se definen como un conjunto finito y ordenado de elementos del mismo tipo. La propiedad “ordenado” quiere decir que el elemento primero, segundo, tercero, …, enésimo de un *array* puede ser identificado y accedido. Cuando se dice que los elementos de un arreglo son homogéneos, significa que son del mismo tipo de datos. Un arreglo puede estar compuesto de todos sus elementos de tipo letras, otro puede tener todos sus elementos de tipo entero, etc. Los arreglos se conocen como matrices en matemáticas, y tablas en temas financieros (Vázquez, 2012).

Según sus dimensiones, los arreglos se clasifican en:

* **Unidimensionales:** de una dimensión, comparables con una lista de elementos.
* **Multidimensionales**: son arreglos de varias dimensiones o los más comunes son bidimensionales (dos dimensiones o tablas) o arreglos multidimensionales (tridimensionales, por ejemplo).

**Figura 17**  
*Matriz unidimensional con 10 elementos*

Icono

Descripción generada automáticamente con confianza baja

Cada vector está compuesto de un número de elementos ya definidos, esto hace que sea una estructura estática (no puede aumentarse ni reducirse). Cada elemento está referenciado por la posición que ocupa dentro del vector. Estas posiciones son llamadas índices; nótese que empieza a contar desde 0 como muestra la figura 15.

Para referirse un vector se usa un identificador como se hace con las variables o constantes, pero queda entre corchetes **[ ]**, por ejemplo si el vector se llama **lista**, y se quiere acceder al quinto elemento del vector de la figura 17, se debe hacer así **lista[4]**, es de notar que la quinta posición se accede con el índice 4, porque se empezó a contar desde 0.

Para realizar las operaciones de asignación, lectura/escritura sobre un vector, se recomienda usar las estructuras repetitivas para, a través de ellas, manipular los índices correspondientes a las posiciones del arreglo.

|  |
| --- |
| **Ejemplo: hacer un programa que pida 5 números enteros al usuario y los guarde en un arreglo de números en un arreglo, luego que imprima los números mayores de 100.** |

**Análisis**

**Formular el problema**: pedirle al usuario 5 números enteros almacenarlos en un arreglo y luego recorrer el arreglo para buscar los números mayores a 100.

**Resultados esperados**: imprimir en pantalla de los 5 números ingresados por el usuario solamente los mayores a 100.

**Datos disponibles**: 5 números ingresados por el usuario.

**Datos adicionales**: por cada uno de esos números validar si son mayores que 100 y mostrar los que lo sean en la pantalla.

**Restricciones**: ninguna.

**Procesos necesarios**: se requiere definir un arreglo de tamaño 5, se necesita pedir 5 veces un entero y almacenarlos en el vector. Luego se debe recorrer las 5 posiciones del vector para comparar en cada uno de los datos si es mayor que 100, si el número comparador es lo que se debe imprimir. Como 5 es un número invariante, el programa lo define como una constante de nombre **tamano** (no se usa la ñ porque no permite ni acentos latinos ni tildes), se debe usar la sentencia repetitiva PARA porque se sabe el número de veces que se debe repetir cada acción.

**Diseño**

**Figura 18**  
*Vector de 5 elementos*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO Vector de 5 enteros;**  **VAR**  **ENTERO** tamano**;**  **ENTERO** lista[tamano]**;**  **INICIO**  **tamano = 5;**  **PARA** i **DESDE** 0 **HASTA** tamano -1  **ESCRIBIR(** "Escriba el número" **);**  **LEER(** lista[ i ] **);**  **FINPARA**  **PARA** i **DESDE** 0 **HASTA** tamano -1  **SI (** lista[ i ] > 100 **)**  **ESCRIBIR(**lista[ i ] **);**  **FINSI**  **FINPARA**  **FIN** |
|  |

1. **Matrices**

Las matrices son arreglos de más de una dimensión, en la siguiente figura se presenta un ejemplo de una matriz de 2 dimensiones que son 4x5 (correspondiente a 4 columnas y 5 filas).

**Figura 19**  
*Matriz de 4 x 5*

|  |  |  |  |
| --- | --- | --- | --- |
| **0,0** | **0,1** | **0,2** | **0,3** |
| **1,0** | **1,1** | **1,2** | **1,3** |
| **2,0** | **2,1** | **2,2** | **2,3** |
| **3,0** | **3,1** | **3,2** | **3,3** |
| **4,0** | **4,1** | **4,2** | **4,3** |

Lo anterior cuando se quiere acceder a un elemento del vector si el identificador usa el corchete doble. Por ejemplo, cuando para acceder al último elemento de la figura 19, usado es matriz seria **matriz[4][3]**;

Para recorrer estas estructuras de datos se requieren ciclos anidados; estas técnicas se abordarán más adelante.

1. **Programación modular**

Existe un área que se denomina ingeniería del *software*, que se interesa en el proceso de creación o producción de software y concentra sus esfuerzos en aportar herramientas, procedimientos y técnicas para su construcción.

La ingeniería del *software* propone que la construcción de *software* esté pensada en la descomposición de un problema en un conjunto de subproblemas independientes entre sí, más sencillos de analizar y resolver; estos subproblemas pueden ser tratados separadamente unos de otros. Lo anterior consiste, básicamente, en la modularidad del *software*.

Debido a la modularidad se pueden probar los subprogramas o módulos de manera independiente, depurando errores antes de su uso en el programa principal y almacenándose para su posterior reutilización cuantas veces se necesite.

El enfoque básico de modularidad en un algoritmo puede estar compuesto de otros algoritmos que se especializan en una función determinada. Para saber cómo descomponer un algoritmo en pequeños algoritmos, se listan los aspectos que pueden ayudar a definir la división de algoritmos:

1. **Según la funcionalidad:** un algoritmo debe realizar una función específica y no ir más allá de sus responsabilidades. Por ejemplo, si se tiene un módulo que calcula la multiplicación de dos números y devuelve el resultado, no sería adecuado que, además, imprima mensajes en pantalla o haga otra cosa diferente de su propósito.
2. **Según su identificación:** cada módulo tiene un nombre de identificación que permite ejecutarlos y diferenciarlos de otros, es recomendable que al elegir este nombre que sea representativo de la función que realiza, de esa forma resulta intuitivo en el momento de usarlo dentro de un algoritmo.
3. **Según sus datos de entrada:** los módulos comúnmente reciben datos de entrada necesarios para realizar las operaciones requeridas. Se pueden clasificar los módulos de acuerdo con los parámetros que se requieren para obtener los resultados solicitados. Al momento de usar el módulo se tienen que agregar todos los parámetros necesarios, en el orden establecido y deben ser del tipo correcto.
4. **Parámetro de salida:** también se puede emplear los resultados a la salida del algoritmo para clasificar los módulos, ya que en el momento de definir un algoritmo se debe indicar el tipo de dato que devuelve al momento de la ejecución del algoritmo.
5. **Pruebas de escritorios:** como se mencionó antes, las pruebas escritorio sirven para depurar y probar que un algoritmo solucione el problema, en esta sesión se mencionarán los objetivos principales y se ejecutará una prueba a un algoritmo de mejor complejidad para su comprensión.

|  |
| --- |
| **Ejemplo: realizar un algoritmo que solicite dos números e imprima cuál es el mayor de ambos, si son iguales, mostrar cualquiera.** |

**Análisis**

**Formular el problema**: es un algoritmo que solita y compara dos números para mostrar el mayor.

**Resultados esperados**: el número que es mayor de ambos.

**Datos disponibles**: 2 números ingresados por el usuario.

**Datos adicionales**: ninguno.

**Restricciones**: validar cuál número es mayor para saber cuál mostrar.

**Procesos necesarios**: se requiere definir los datos a recibir, pedirle que al usuario que digite los números, comparar para saber cuál es el mayor y mostrarlo.

**Diseño**

**Figura 20**  
*Algoritmo el mayor de dos números*

|  |  |
| --- | --- |
| **Diagrama  Descripción generada automáticamente** | **ALGORITMO** El mayor de dos números**;**  **VAR**  **ENTERO** numero1**;**  **ENTERO** numero2**;**  **INICIO**  **ESCRIBIR(** "Escriba un número" **);**  **LEER(** numero1 **);**  **ESCRIBIR(** "Escriba un número" **);**  **LEER(** numero2 **);**  **SI** numero1 > numero2  **ESCRIBIR(** numero1 **);**  **SINO**  **ESCRIBIR(** numero2 **);**  **FINSI**  **FIN** |
|  |

**Diseño de prueba 1**

Selección de datos: se usarán para la prueba los números 34 y 16.

Salida esperada: debe mostrar el número 34.

**Figura 21**  
*Prueba de escritorio de algoritmo el mayor de dos números*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. ESCRIBIR("Escriba en número");  |  |  |  | | --- | --- | --- | | numero1 | numero2 | Salida en pantalla | |  |  | Escriba un número | |  |  | |  |  | | 1. LEER( numero1 );  |  |  |  | | --- | --- | --- | | numero1 | numero2 | Salida en pantalla | | 34 |  | Escriba un número  34 | |  |  | |  |  | |
| 1. ESCRIBIR("Escriba en número");  |  |  |  | | --- | --- | --- | | numero1 | numero2 | Salida en pantalla | | 34 |  | Escriba un número  34  Escriba un número | |  |  | |  |  | | 1. LEER( numero1 );  |  |  |  | | --- | --- | --- | | numero1 | numero2 | Salida en pantalla | | 34 |  | Escriba un número  34  Escriba un número  16 | |  | **16** | |  |  | |  |  | |
| 1. SI  numero1 > numero2  |  |  |  | | --- | --- | --- | | numero1 | numero2 | Salida en pantalla | | 34 |  | Escriba un número  34  Escriba un número  16 | |  | **16** | |  |  | |  |  | | 1. ESCRIBIR( numero1 );  |  |  |  | | --- | --- | --- | | numero1 | numero2 | Salida en pantalla | | 34 |  | Escriba un número  34  Escriba un número  16  34 | |  | **16** | |  |  | |  |  | |  |  | |

Una prueba de escritorio no es determinante en este ejercicio, es necesario hacerla para los dos casos faltantes, en caso que el segundo número es mayor que el primero y el caso en que ambos números ingresados son iguales. Se propone que use el ejemplo de la figura 18 para realizar el desarrollo final del ejemplo.