2019

Cuadernillo de Programación Taller

MARIA ELISA RAMIREZ – AGUSTINA AVILÉS

El proceso de programación

Elaborar un programa de computadora implica llevar a cabo, una serie de pasos secuenciales y cronológicos que comienza con la detección y definición del problema y conducen a la implementación del programa que lo soluciona. A continuación se describen los pasos a seguir.

1. Definición del problema

Este proceso inicia cuando surge la necesidad de resolver algún problema mediante la computadora. Para empezar, se debe de identificar el problema y comprender la utilidad de la solución que se alcance. Es menester tener una visión general del problema estableciendo las condiciones iniciales y, además, los límites del problema; es decir, donde empieza y donde termina. Por ejemplo, si tenemos que calcular el sueldo de un empleado, la solución que se logre permitirá obtener la cantidad que debe pagársele. El problema anterior consiste en un pago de sueldos, y parte de que cada empleado tiene algunos atributos como su nombre, la antigüedad y el sueldo que percibe por cada hora trabajada.

2. Análisis del problema

A continuación es necesario entender en detalle el problema en cuestión, para obtener una radiografía del mismo en términos de los DATOS disponibles como materia prima, y definir el PROCESO necesario para convertir los datos en la INFORMACIÓN requerida.

La primera etapa consiste en definir los resultados esperados, es decir, la información que deberá producirse como salida. Respecto al problema de pago de salarios tenemos que se requiere la siguiente salida:

- Nombre del empleado: xxxxxxxxxxxxx
- Sueldo: 99999999

La segunda etapa consiste en identificar los DATOS que se tienen como materia prima y que constituirán la entrada del programa. En este ejemplo tenemos:

- El nombre del empleado
- El número de horas trabajadas
- La cuota por hora

La tercera etapa tiene como finalidad determinar el PROCESO necesario para convertir los datos de entrada en la información que se tendrá como salida. Volviendo al ejemplo, puesto que se requieren dos datos de salida, determinemos el proceso de la siguiente manera:

a. ¿Cómo se calcula el nombre del empleado?

No implica ningún cálculo, pues es un dato que se obtiene como entrada y que no se modifica.

b. ¿Cómo se calcula el sueldo?

El sueldo es un dato que no existe como entrada, pero se obtiene o genera multiplicando las horas trabajadas por la cuota horaria. En este momento ya se tiene una comprensión clara del problema, y podemos avanzar hacia el siguiente paso

3. Diseño del programa

Durante este paso se procede a diseñar la lógica para la solución del problema, haciendo dos cosas:

a) Elaborar el algoritmo: Se diseña el algoritmo de la solución al problema; es decir, se estructura la secuencia lógica y cronológica de los pasos que la computadora deberá seguir, utilizando alguna técnica convencional como el pseudocódigo, los diagramas de flujo, los diagramas de llaves, etc. Equiparando esta actividad con la construcción de una casa, equivale a diseñar el plano arquitectónico de la misma.
b) Pruebas de escritorio o traza: Se simula el funcionamiento del algoritmo con datos propios respecto del problema, y se comprueban a mano los resultados a fin de validar la correcta operación del algoritmo. Si quedamos satisfechos con los resultados de la prueba habremos agotado este punto, pero en caso contrario se deberá modificar el algoritmo y posteriormente volverlo a probar hasta que éste correcto. Es posible que se deba retroceder a cualquier paso precedente.

En este momento se tiene ya diseñada la solución al problema, y estamos listos para pasar al siguiente punto.

4. Codificación del problema

En este paso se procede a codificar el programa en el lenguaje de programación que vayamos a utilizar. Este proceso es sumamente sencillo ya que tenemos diseñado el programa, solo nos centramos en convertir las acciones del algoritmo en instrucciones de computadora. El programa codificado debe editarse, compilarse, probarse y depurarse, es decir, se ejecuta para verificar su buen funcionamiento y se hacen las correcciones o los ajustes pertinentes hasta que esté correcto.

Para que un programa pueda ser entendido y ejecutado por la computadora, debe estar en lenguaje máquina para ello se debe producir el proceso de compilación, este proceso consiste en lo siguiente: una vez que tenemos codificado el programa en papel, debe ser introducido mediante el procese de edición, para lo cual se utiliza un editor que nos permite crear un archivo en el cual introducimos el programa, creándose el programa fuente con las instrucciones que nosotros elaboramos

en el lenguaje que estemos utilizando en este momento. El programa fuente es sometido al proceso de compilación, que mediante un compilador se traduce instrucción por instrucción a código objeto, creándose un archivo con el programa objeto el cual es entendible directamente por la máquina. Si en el proceso se encuentra algún error, se suspende el proceso; el programador debe corregir el error en el programa fuente y luego someterlo de nuevo al proceso de compilación.

Una vez que el proceso de compilación ha terminado con éxito, se tiene el programa objeto, el cual puede ser ejecutado por la computadora que seguirá las instrucciones paso a paso llevando a cabo las acciones que se indican emitiendo los resultados correspondientes, si estos no son satisfactorios o existen errores, el proceso de programación debe ser repetido desde alguno de los pasos precedentes

5. Implementación del programa

Una vez que el programa está correcto, se instala y se pone a funcionar. Debe ser supervisado continuamente para detectar posibles cambios o ajustes que sean necesarios realizar

6. Mantenimiento del programa

Un programa que está en operación, por un lado podría presentar errores, los cuales deben corregirse; por otro lado podría requerir cambios o ajustes en sus datos, proceso o información; esto implica que eventualmente necesitará mantenimiento para adecuarlo a los cambios de la empresa. Esto nos sitúa en una dinámica infinita, ya que si surge la necesidad de darle mantenimiento tendremos que regresar a algún paso precedente; al 4, al 3, al 2 o al punto 1 para definir de nuevo el problema.

Algoritmo

En el proceso de programación hay un paso que es crucial a la hora de desarrollar un programa; es el diseño del programa, en otras palabras, diseñar o elaborar el algoritmo de la solución.

El algoritmo es una secuencia ordenada y cronológica de pasos que llevan a la solución de un problema o a la ejecución de una tarea

Los pasos del algoritmo deben tener las siguientes características:

- Ser simples, claros, precisos, exactos
- Tener un orden lógico
- Tener un principio y un fin

Un algoritmo debe producir un resultado en un tiempo finito. Los métodos que utilizan algoritmos se denominan métodos algorítmicos, en oposición a los métodos que implican algún juicio o interpretación que se denominan métodos heurísticos. Los métodos algorítmicos se pueden implementar en computadoras; sin embargo, los procesos heurísticos no han sido convertidos fácilmente en las computadoras. En los últimos años las técnicas de inteligencia artificial han hecho posible la implementación del proceso heurística en computadoras.

Ejemplos de algoritmos son: instrucciones para andar en una bicicleta, hacer una receta de cocina, las instrucciones para armar o utilizar un juguete, obtener el mayor de 5 números dado. Los algoritmos se pueden expresar en Pseudocódigo, diagramas de flujo o diagramas de llaves.

Por ejemplo:

- Elaborar un algoritmo para que guie a una persona a cambiar una lamparita quemada Los pasos de un algoritmo para cambiar una lamparita podrían ser:
    1. Quitar el foco fundido
    2. Colocar el foco nuevo
    3. Fin

Es probable que no todos pensemos exactamente los mismos pasos para cambiar la lamparita, pero si conducen de manera efectiva a realizar la tarea entonces estará en lo correcto. Si bien es cierto que son pocos pasos, si se entienden y se logra cambiar el foco, entonces estará correcto.

Ahora bien, supongamos que estamos tratando de entrenar a un robot para que haga la tarea; en tal caso no funcionará el algoritmo, tendremos que ser más específicos y claros tomando en cuenta las capacidades elementales del robot.

El algoritmo más detallado sería:

1. Colocar la escalera
2. Subir a la escalera
3. Quitar la lamparita quemada
4. Bajarse de la escalera
5. Obtener la lamparita de repuesto
6. Subir a la escalera
7. Colocar la lamparita de repuesto
8. Bajar de la escalera
9. Guardar la escalera

10. Fin

- Diseñar un algoritmo para volver desde el cine, en función del dinero que tengo:
    1. ¿Tengo bastante dinero? Si paso 2 No paso 5
    2. Buscar un taxi
    3. ¿Lo encontré? Si paso 4 No paso 2
    4. Tomar taxi e ir al paso 6
    5. ¿Tengo dinero para el autobús? Si paso 6 No paso 7
    6. Tomar el autobús e ir al paso 8
    7. Volver caminando
    8. Fin

Todas las actividades que llevamos a cabo los seres humanos son algoritmos que hemos aprendido a seguir. Caminar y lavarse los dientes, por ejemplo, son secuencias lógicas de pasos. La civilización está basada en el orden de las cosas y de acciones; estas se organizan conforme a secuencias lógicas, y a esto se le llama programación.

Durante el desarrollo de algoritmos de computadora, es necesario idear los pasos que la máquina deberá seguir para realizar alguna tarea, actividad o resolver algún problema de nuestra vida cotidiana. Cuando se presenta el problema de realizar cierta tarea, debemos efectuar ciertas actividades subyacentes tales como hacer preguntas, buscar objetos conocidos o dividir el problema en partes. A menudo hacemos todo eso inconscientemente, pero al elaborar algoritmos para solucionar problemas con la computadora debemos detallar esas actividades de manera explícita.

Si se nos solicita una tarea verbalmente, por lo general hacemos preguntas hasta que quede claro lo que debemos hacer.

Algunas preguntas comunes en el contexto de la programación son:

- ¿Qué datos tenemos para trabajar?, y ¿Cuál es su apariencia?
- ¿Cuántos datos hay?
- ¿Cómo sabemos que ya están procesados todos los datos?
- ¿Cuál debe ser el formato de la salida?
- ¿Cuántas veces debe repetirse el proceso?
- ¿Qué condiciones especiales de error pueden presentarse?

Ya sabemos lo que es un algoritmo y conocemos la manera de diseñar soluciones para problemas de la vida cotidiana. Ahora aplicaremos estos conceptos a la elaboración de algoritmos para programar computadoras, utilizando Diagramas de llaves o Pseudocódigo.

Estas dos son técnicas para diseñar algoritmos de programas que permiten definir las estructuras de datos, las operaciones que se aplicaran a los datos y la lógica que tendrá el programa de computadora para solucionar un determinado problema

El ser humano realiza los razonamientos a partir de información adquirida que está almacenada en su cerebro. También el ordenador tiene que almacenar en su interior toda la información necesaria para realizar el proceso automático que le pedimos que ejecute.

La parte del ordenador que realiza esta función es la Memoria principal, interna o central. Es la denominada memoria R.A.M. (memoria de acceso directo).

La memoria está constituida por multitud de posiciones de memoria (celdas de memoria) numeradas de forma consecutiva, capaces de retener elementos de información que vamos a denominar datos o valores.

También se almacenarán las acciones (instrucciones) que se estudiarán más adelante.

Los algoritmos ya sea realizados en pseudocódigo o con Diagramas de Llaves deben contener los siguientes elementos básicos: las variables, las operaciones primitivas elementales y las estructuras de control. A continuación explicaremos cada una de ellas.

Variables

En el programa hay datos que van cambiando durante el desarrollo del mismo, para representar y manejar estos datos se utilizan lo que se denomina VARIABLE. Las variables se encuentran en direcciones de memoria y contienen valores, se identifican por un nombre y el tipo de dato que indica los valores que puede contener

Todo dato que vaya a ser introducido a la computadora, y todo dato que vaya a ser generado o calculado a partir de otros datos para obtener algún resultado, debe identificarse y manejarse en forma de variable.

En resumen una variable tiene las siguientes características:

- Nombre: es el identificador de la variable y que servirá para referenciarla.

Como buena práctica de programación se aconseja que los nombre de las variables tengan una relación con la información que van a almacenar, de modo que resultaría lógico usar la variable llamada TotalFactura para representar el valor total de una factura de venta. También es aconsejable que los nombres de variables

formados por varias palabras, usen mayúsculas en la primera letra de cada palabra para distinguirlas, tal como se ha hecho con la variable TotalFactura. Con esto haremos que las escrituras de nuestros programas resulten más fáciles de entender al momento de realizarle futuras modificaciones.

El nombre de una variable o una constante puede tener una longitud máxima de 127 caracteres y el primer carácter debe ser una letra, los caracteres siguientes pueden ser letras, dígitos o el carácter de subrayado (_). No admiten espacios en blanco, ni caracteres especiales o acentuados (%, $, &, @, !, á, ñ, Ñ, ...). Tampoco se permiten usar nombres repetidos, ni usar palabras reservadas del lenguaje (Begin, If, For, While, End, Until, ...). Turbo Pascal no distingue entre mayúsculas y minúsculas, así por ejemplo, las variables TotalFactura y totalfactura serían consideraras como la misma variable.

- Tipo de dato: Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como `b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.
- Operaciones primitivas elementales: asignación, lectura y escritura.

Operaciones primitivas elementales

Ya hemos visto que una variable está relacionada con posiciones de memoria que van a contener valores que cambiarán durante la ejecución del programa, por tanto es necesario tener a nuestra disposición una acción que nos permita dar los valores adecuados a cada variable para obtener al final del proceso los resultados correctos. La instrucción que permite asociar un valor a una variable se denomina ASIGNACIÓN.

Debemos tener en cuenta el tipo de variable para no cometer errores a la hora de la asignación, es decir, nunca se permitirá asignar por ejemplo, a una variable numérica el valor "a" que ya sabemos es de tipo carácter, o a una variable de tipo cadena el valor 124.56 que es de tipo numérico.

Recordaremos también que la asignación de un valor a una variable supone una escritura en memoria y por tanto una operación destructiva del valor que tuviera la variable anteriormente. El símbolo utilizado para indicar esta acción de asignación será "<------", de forma que la operación se señalará de la siguiente manera:

variable <------ valor o expresión.

A←VALOR

Entrada de datos - instrucción leer

Las instrucciones de entrada permiten asociar un valor a una variable, pero el valor es tomado del exterior por medio de un dispositivo de entrada: por ejemplo el teclado.

Leer NV

Donde NV es el nombre de la variable.

Tras la operación de lectura, la variable NV contiene el valor dado por el usuario a través de un dispositivo de entrada que generalmente es el teclado

El tipo de dato suministrado desde el exterior debe ser compatible con el tipo de la variable NV.

Entrada de datos - instrucción escribir

Las instrucciones de salida permiten mostrar los resultados obtenidos al exterior, sin ésta instrucción las soluciones no podrían darse a conocer. Mediante la operación de salida se transfiere el valor de una expresión a un dispositivo de salida generalmente el monitor.

Sintaxis:

ESCRIBIR ('El resultado es' , NV)

Se suele utilizar también para mostrar mensajes informativos al usuario que estarán delimitados por comillas

ESCRIBIR ('Hola')

Ejemplos:

- Leer un número y mostrarlo:
- Leer Numero Escribir Numero
- Lee nombre, apellido y DNI y saludar con Hola y el nombre de la persona. Leer Nombre, Apellido, DNI
- Escribir 'Hola' , Nombre.

Estructuras de control

Las estructuras de control controlan el flujo de ejecución de un programa o función.

Las instrucciones o sentencias se organizan en tres tipos de estructuras de control que sirven para controlar el flujo de ejecución.

1. Secuenciales: las instrucciones se ejecutan sucesivamente una debajo de la otra.
2. Selectivas: permiten elegir entre dos alternativas dependiendo de la condición.
3. Repetitivas: una serie de instrucciones que se repiten una y otra vez hasta que se cumple cierta condición.

Estructuras de control secuenciales

Las instrucciones se siguen en una secuencia fija que normalmente viene dada por el número de renglón. Es decir que las instrucciones se ejecutan de arriba hacia abajo.

Ejemplo

- Algoritmo que suma dos valores cualesquiera sean.

Pseudocódigo

```
Algoritmo sumaNumeros

    Escribir 'Ingresar el primer valor'
    Leer numero1

    Escribir 'Ingresar el segundo valor'
    Leer numero2

    Suma <- numero1 + numero2
    Escribir 'El valor de la suma es: ', Suma

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main(){

    int numero1, numero2, suma;

    printf ("Ingresar el primer valor");
    scanf ("%d", &numero1);

    printf ("Ingresar el segundo valor");
    scanf ("%d", &numero2);

    suma = numero1 + numero2;

    printf ("El valor de la suma es: %d", suma);
}
```

Pseudocódigo

El pseudocódigo es otro lenguaje artificial e informal útil para el desarrollo de algoritmos. No es un lenguaje de programación verdadero y, por lo tanto, no puede ser compilado y ejecutado.

En pseudocódigo se describen los algoritmos utilizando una mezcla de lenguaje común, con instrucciones de programación, palabras claves, etc. El objetivo es que el programador se centre en la solución lógica del algoritmo y no en la implementación en un lenguaje de programación concreto (con las posibles complicaciones en las reglas sintácticas), o, en otras palabras, ayudan a "pensar" un programa antes de escribirlo en un lenguaje de programación formal.

Características y partes del pseudocódigo

Las principales características de este lenguaje son:

1. Es una forma de representación sencilla de utilizar y de manipular.
2. Facilita el paso del programa al lenguaje de programación.
3. Es independiente del lenguaje de programación que se vaya a utilizar.
4. Es un método que facilita la programación y solución al algoritmo del programa. Todo documento en pseudocódigo debe permitir la descripción de:
    i. Instrucciones primitivas.
    ii. Instrucciones de proceso.
    iii. Instrucciones de control.
    iv. Instrucciones compuestas.
    v. Instrucciones de descripción.

Estructura a seguir en su realización:

1. Cabecera.
2. Nombre del Programa.
3. Variables.
4. Cuerpo.
5. Inicio.
6. Instrucciones.
7. Fin.

Las operaciones primitivas Leer y Escribir son iguales, la operación de asignación se representa con el signo =.

- Escribir un algoritmo que calcule la superficie de un triángulo en función de la base y la altura.

Pseudocódigo

```
Algoritmo calcularSuperficie

    Escribir 'Ingresa el valor de la base'
    Leer BASE

    Escribir 'Ingresar el valor de la altura'
    Leer ALTURA

    SUPERFICIE <-(BASE * ALTURA)/2
    Escribir 'La superficie es: ', SUPERFICIE

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main(){

    int base, altura, superficie;

    printf( "Ingrese el valor de la base ");
    scanf ("%d", &base);

    printf( "Ingrese el valor de la altura ");
    scanf ("%d", &altura);

    superficie = (base*altura)/2;

    printf("La superficie es: %d", superficie);
}
```

Estructuras de control de selección o condicionales

La estructura condicional permite bifurcar el "flujo "del programa, es decir que se desarrollen un conjunto de instrucciones dentro de un algoritmo, acorde al resultado de la evaluación de una proposición que llamaremos la condición; disponemos de tres estructuras alternativas diferentes: alternativa simple, alternativa doble y alternativa múltiple.

Estructura de control selectiva simple

Están compuestas únicamente de una condición únicamente de una condición. La estructura SI – ENTONCES evalúa la condición y en tal caso si la condición es verdadera, entonces ejecuta la acción SI (o acciones si son varias). Si la condición es falsa, entonces no hace nada. La condición puede ser cualquier expresión lógica que de cómo resultado verdadero o falso

Para armar las condiciones se utilizan los operadores relacionales, y para armar condiciones compuestas se utilizarán los operadores o conectivos AND o CONJUNCIÓN y OR o DISYUNCIÒN, para determinar el valor de verdad de las expresiones se utilizarán las respectivas tablas de verdad.

PSEUDOCÓDIGO	C	SIGNIFICADO
OPERADORES ARITMÉTICOS		
+	+	Suma
–	–	Resta
*****	*	producto
MOD	%	Resto división entera
DIV	/	Cociente división entera
OPERADORES RELACIONALES		
>	>	Mayor
<	<	Menor
>=	>=	Mayor igual
>=	>=	Menor igual
=	==	Igual
< >	!=	distinto
OPERADORES LÓGICOS		
AND	&&	Conjunción (y)
OR	||	Disyunción (o)
¬	!	Negación

Ejemplo

- Diseñar un algoritmo que permita calcular la raíz cuadrada de un número, si el número ingresado es negativo se debe convertir en positivo antes de calcular la raíz cuadrada.

Pseudocódigo

```
Algoritmo calcularRaizCuadrada

    Escribir 'Ingresar el valor de un numero'
    Leer num

    Si (num < 0) Entonces
        num<-num * (-1)
    FinSi

    resultado_raiz<-raiz(num)
    Escribir 'El valor de la raiz es', resultado_raiz

FinAlgoritmo
```

C

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

int main(){

    int num;
    float resultado_raiz;

    printf ("Ingrese numero");
    scanf ("%d", &num);

    if (num < 0)
        num = num * (-1);

    resultado_raiz = sqrt(num);

    printf("La raiz es: %f", resultado_raiz);

}
```

Estructura de control selectiva doble

La estructura selectiva doble es semejante a la simple, la diferencia radica en que la doble permite especificar un conjunto de instrucciones a desarrollar en caso que el valor de la condición sea verdadero y otro conjunto distinto de instrucciones en caso que el valor de la condición sea falso

SI (condición) ENTONCES

Instrucción1

instrucción2

.....

instrucciónN

SINO

Instrucción3

FINSI

Ejemplo

- Diseñar un algoritmo que permite ingresar el monto dinero que debería pagar un cliente de una tienda y calcular el descuento que se le hará sobre este monto considerando que por encima de $1000 se realizará un descuento es del 30% y si el monto de compra es de $1000 o más el descuento que se le hará será del 20%. Se debe calcular y mostrar cuanto debe pagar el cliente.

Pseudocódigo

```
Algoritmo calcularDescuento

    Escribir 'Ingresar el monto comprado'
    Leer monto

    Si (monto >1000) Entonces
        descuento <-(monto* 30)/100
    sino
        descuento <-(monto* 20)/100
    Fin Si

    montoF <- monto - descuento
    Escribir 'El total a pagar es', montoF

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main (){

    float monto, descuento, montoF;

    printf ("Ingrese monto comprado");
    scanf ("%f", &monto);

    if (monto > 1000)
        descuento = (monto*30)/100;
    else
        descuento = (monto*20)/100;

    montoF = monto - descuento;

    printf ("El total a pagar es: %f", montoF);

}
```

- Diseñar un algoritmo que permita leer dos números enteros e indique cual es el mayor de ambos.

Pseudocódigo

```
Algoritmo buscarMayor

    Escribir 'Ingresar un número'
    Leer numeroA

    Escribir 'Ingresar otro número'
    Leer numeroB

    Si (numeroA > numeroB) entonces
        numMayor <- numeroA
    SiNo
        numeroMayor <- numeroB
    FinSi

    Escribir "El mayor es: ", numMayor

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main(){

    int numeroA, numeroB, numeroMayor;

    printf("Ingresar numero");
    scanf("%d", &numeroA);

    printf("Ingresar otro numero");
    scanf("%d", &numeroB);

    if (numeroA > numeroB)
        numeroMayor = numeroA;
    else
        numeroMayor = numeroB;

    printf("El numero mayor es: %d", numeroMayor);

}
```

Estructuras condicionales múltiples

Hasta ahora las estructuras vistas (simples y doble) solo nos permiten tomar una o dos decisiones, aunque la estructura múltiple ayuda a elegir varias, opciones no siempre es la adecuada para todas las aplicaciones. La Estructura de decisión múltiples evaluara una expresión que podrá tomar n caminos y solo se realizara una de las n acciones o lo que es igual, el flujo del algoritmo seguirá solo un determinado camino ente los n posibles.

SI (condición1)

```
        Instrucción1
SINO
        SI (condición2)
                Instrucción2
        SINO
                SI (condición3)
                        Instrucción3
                        .
                        .
                SINO
                        InstrucciónDefecto
                FINSI
        FINSI
FINSI
```

Ejemplo

- Diseñar un algoritmo que permita solucionar el siguiente problema: En un club los socios pagan diferentes cuotas dependiendo de la categoría a la que pertenecen. Si son de categoría 1 pagan una cuota de $100; si son de categoría 2 pagaran una cuota de $150, si son de categoría 3 pagaran una cuota de $200 y si son de categoría 4 pagaran $250. Se debe mostrar la cuota que debe pagar el socio.

Pseudocódigo

```
Algoritmo cuotaCategoria

    Escribir 'Ingresar la categoria'
    Leer categoria

    Si (categoria = 1) Entonces
        Escribir ' la cuota es de $100'
    Sino
        Si (categoria = 2) Entonces
            Escribir ' la cuota es de $150'
        Sino
            si (categoria = 3) Entonces
                Escribir ' la cuota es de $200'
            sino
                Escribir ' la cuota es de $250'
            FinSi
        Finsi
    FinSi

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main(){

    int categoria;

    printf ("Ingrese categoria");
    scanf ("%d", &categoria);

    if (categoria == 1)
        printf("la cuota es de $100");
    else if(categoria == 2 )
            printf("la cuota es de $150");
        else if (categoria == 3)
            printf("la cuota es de $200");
            else
            printf("la cuota es de $250");

}
```

Estructuras de control repetitivas

Cuando escribimos un algoritmo las instrucciones se irán ejecutando por orden, pero en determinadas ocasiones es necesario que esto no suceda así, necesitaremos que, dependiendo de la situación, una misma instrucción se ejecute varias veces, para esto se necesita una estructura de control repetitiva.

A estas sentencias se les da el nombre de Bucles, en todos los lenguajes de programación se utiliza este tipo de estructuras de control, en el lenguaje Pascal se utilizan las siguientes:

- REPETIR MIENTRAS
- REPETIR .....HASTA
- PARA

Asociadas a los bucles se encuentran a menudo algunas variables auxiliares. Como siempre se utilizan de la misma manera, las llamamos con un nombre propio (contador, acumulador, etc.), pero hay que dejar claro que no son más que variables comunes, aunque se usan de un modo especial.

Acumuladores

Las variables acumuladoras tienen la misión de almacenar resultados sucesivos, es decir, de acumular resultados, de ahí su nombre.

Las variables acumuladoras también deben ser inicializadas. Si llamamos "acum" a un acumulador, escribiremos antes de iniciar el bucle algo como esto:

acum = 0

Por supuesto, el valor inicial puede cambiar, dependiendo de la naturaleza del problema. Más tarde, en el cuerpo del bucle, la forma en la que nos la solemos encontrar es:

acum = acum + N

...siendo N otra variable. Si esta instrucción va seguida de otras:

acum = acum + M

acum = acum + P

... estaremos acumulando en la variable "acum" los valores de las variables M, N, P, etc, lo cual resulta a veces muy útil para resolver ciertos problemas repetitivos.

En este algoritmo, cont es una variable contador típica de bucle. Se ha usado un bucle "para", que es lo más sencillo cuando conocemos previamente el número de repeticiones (10 en este caso). La variable Nsuma es el acumulador, donde se van sumando los diferentes valores que toma N en cada repetición. se usa para cada uno de los números introducidos por el teclado, y la variable

Observe como, al principio del algoritmo, se le asigna al acumulador el valor 0. Esta es una precaución importante que se debe tomar siempre porque el valor que tenga una variable que no haya sido usada antes es desconocido (no tiene por qué ser 0).

Variables Contadoras

Un contador es una variable (casi siempre de tipo entero) cuyo valor se incrementa o decrementa en cada repetición de un bucle en una cantidad fija. Es habitual llamar a esta variable "cont" (de contador) o "i" (de índice).

El contador suele usarse de este modo:

Primero se inicializa antes de que comience el bucle. Es decir, se le da un valor inicial. Por ejemplo:

cont← 1

Segundo, se modifica dentro del cuerpo del bucle. Lo más habitual es que se incremente su valor en una unidad. Por ejemplo:

cont ← cont + 1

Esto quiere decir que el valor de la variable "cont" se incrementa en una unidad y es asignado de nuevo a la variable contador. Es decir, si cont valía 1 antes de esta instrucción, cont valdrá 2 después.

Otra forma típica del contador es: cont = cont – 1

En este caso, la variable se decrementa en una unidad; si cont valía 1 antes de la instrucción, tendremos que cont valdrá 0 después de su ejecución.

El incremento o decremento no tiene por qué ser de una unidad. La cantidad que haya que incrementar o decrementar vendrá dada por la naturaleza del problema.

Tercero, se utiliza en la condición de salida del bucle. Normalmente, se compara con el valor máximo (o mínimo), que debe alcanzar el contador para dejar de repetir las instrucciones del bucle.

El uso de contadores es casi obligado en bucles que deben ejecutarse un determinado número de veces. Recuerde que siempre hay que asignar al contador un valor inicial para la primera ejecución del bucle (cont ← 1 por ejemplo) e ir incrementándolo (o decrementándolo, según el algoritmo) en cada repetición con una instrucción del tipo cont ← cont + 1 en el cuerpo del bucle. De lo contrario habremos escrito un bucle infinito.

Por último, hay que prestar atención a la condición de salida, que debe estar asociada al valor del contador en la última repetición del bucle. Mucho cuidado con el operador relacional (<, >, <=, >=, etc) que usemos, porque el bucle se puede ejecutar más o menos veces de lo previsto.

Repetir mientras

Esta estructura de control indica que se ejecuten una o más sentencias mientras se cumpla una determinada condición. La condición se forma utilizando los operadores relacionales cuya evaluación tendrán como resultado un valor de verdad, es decir, VERDADERO o FALSO.

Esta sentencia comprueba la condición al comienzo del bucle La condición tiene que ser verdadera para que se ejecuten las sentencias que están dentro del bucle y finalizará cuando la condición sea falsa.

Dentro del bucle debe existir, por lo menos, una sentencia que modifique el valor de la variable o expresión que forma parte de la condición, de lo contrario se puede producir una situación denominada BUCLE INFINITO, ya que este nunca finalizaría.

Si la condición es falsa al comenzar el bucle, este no ejecutará nunca las sentencias que están dentro del mismo.

Ejemplos

- Diseñar un programa que imprima números del 1 al 10

Pseudocódigo

```
1  Algoritmo imprimirNumeros
2
3      cont=1
4
5      Mientras (cont <= 10) Hacer
6
7          Escribir cont
8          cont=cont+1
9
10     Fin Mientras
11
12 FinAlgoritmo
```

C

```
1  #include <stdio.h>
2
3  int main(){
4
5      int cont = 1;
6
7      while (cont <= 10){
8
9          printf ("%d", cont);
10         cont++;
11
12     }
13 }
```

- Diseñar un programa que imprima números pares del 10 al 30.

Pseudocódigo

```
Algoritmo imprimirNumeros

    cont=10

    Mientras (cont <= 30) Hacer

        Escribir cont
        cont=cont+2

    Fin Mientras

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main(){

    int cont = 10;

    while (cont <= 30){

        printf ("%d", cont);
        cont = cont + 2;

    }
}
```

Repetir hasta

La diferencia principal respecto al bucle "REPETIR MIENTRAS" es que el bucle "REPETIR MIENTRAS" " lo primero que hace es la comparación de la condición, por tanto, puede que el bucle "REPETIR MIENTRAS" no se ejecute ninguna vez si la condición inicial es falsa, mientras que el bucle "REPETIR HASTA" lo primero que hace es ejecutar la sentencia o sentencias, por tanto, el bucle "REPETIR HASTA", siempre se ejecuta una vez al menos.

El cuerpo del bucle se repite hasta que la condición sea verdadera repetir

Ejemplos

- Diseñar un programa que permita mostrar los 10 primeros números enteros positivos.

Pseudocódigo

```
Algoritmo mostrarNum

    num <- 1

    Repetir

        Escribir num
        num <- num +1

    Hasta Que (num = 10)

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main(){

    int num = 1;

    do{

        printf("%d", num);
        num++;

    }while(num <= 10);

}
```

- Diseñar un algoritmo que permita calcular la suma de los n primeros números enteros positivos.

Pseudocódigo

```
Algoritmo sumaNumeros

    Escribir 'Íngrese un numero'
    leer num
    suma <-0
    k<-0

    Repetir
        k<-k + 1
        suma<-suma +k
    Hasta Que (k = num)

    Escribir 'La suma es ',suma

FinAlgoritmo
```

C

```c
#include <stdio.h>

int main (){

    int num, suma = 0, k = 0;

    printf ("Ingrese numero");
    scanf ("%d", &num);

    do{
        k++;
        suma = suma + k;

    }while( k != num);

    printf("La suma es: %d", suma);
}
```

El bucle "PARA" es una sentencia repetitiva cuya principal característica es que el

número de veces que se repite es constante Para ello, la declaración de dicha estructura es de la siguiente forma:

PARA <variable de control> := <valor inicial> HASTA <valor final> HACER

<sentencias>;

La variable de control es la variable que se va a incrementar automáticamente durante la ejecución del bucle para saber la condición en la que debe terminar el bucle.

El valor inicial es el valor con el que se inicia la variable de control y el valor final es con el que se compara para saber la condición de terminación. Las sentencias son las ejecutadas en cada pasada del bucle. Por tanto, un bucle "PARA" O "FOR "es una estructura de Pascal que hace que se repita una sentencia o número de sentencias un número determinado de veces.

Aunque a primera vista pueda resultar más atractivo FOR, existen limitaciones en su aplicación ya que en el bucle FOR siempre se incrementa o decrementa (de uno en uno) los valores de la variable de control de bucle y no de dos en dos o de tres en tres, o con valores fraccionarios.

El número de iteraciones de un bucle FOR siempre es fijo y se conoce de antemano: Valor final - Valor inicial +1.

Ejemplos

- Diseñar un algoritmo que imprima los mil primeros números en orden creciente.

Pseudocódigo

```
1  Algoritmo imprimirNumeros
2
3      Para cont<-1 hasta 1000 Con Paso 1 Hacer
4          Escribir cont
5      FinPara
6
7  FinAlgoritmo
```

C

```
1   #include <stdio.h>
2
3   int main(){
4
5       int cont;
6
7       for(cont = 1; cont <= 10; cont++){
8
9           printf("%d", cont);
10      }
11  }
```

❖ Diseñar un programa que imprima los cuadrados de los primeros 100 números naturales.

Pseudocódigo

```
#include <stdio.h>

int main(){

    int cont;

    for(cont = 1; cont <= 1000; cont++){

        printf("%d", cont);
    }
}
```

C

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

int main(){

    int cont, cuadrado, potencia = 2;

    for(cont = 1; cont <= 100; cont++){

        cuadrado = pow(cont, potencia);
        printf("%d \n", cuadrado);
    }
}
```

Cuándo utilizar Repetir mientras – Repetir hasta - Para

❖ Utilizar la sentencia o estructura PARA cuando se conozca el número de iteraciones, y siempre que la variable de control de bucle sea de tipo ordinal.

❖ Utilizar la estructura REPETIR …HASTA cuando el bucle se realice por lo menos una vez.

❖ En todos los demás casos utilizar la sentencia REPETIR…MIENTRAS.

Ejercicio: De N números leídos contar la cantidad de números pares e impares. Luego sumarlos. Mostrar la cantidad de cada uno y la suma.

- A continuación, plantearemos la solución con las diferentes estructuras repetitivas vistas en clase (en Pseint y su respectiva implementación en C).

REPETIR MIENTRAS

Pseint

```
Algoritmo contarParesImpares
    Escribir "Ingrese cantidad de numeros"
    Leer cantidad

    //inicializo los contadores y acumuladores
    cont <- 1
    pares <- 0
    impares <- 0
    sumaPares <- 0
    sumaImpares <- 0

    Mientras (cont <= cantidad) Hacer
        Escribir "Ingrese numero"
        Leer num
        Si (num MOD 2 = 0) Entonces //si el numero leido es par
            pares <- pares+1 //incremento contador de pares
            sumaPares <- sumaPares+1 //acumulo los numeros pares
        SiNo
            //si el numero leido es impar
            impares <- impares+1 //incremento contador de impares
            sumaImpares <- sumaImpares+1 //acumulo los numeros impares
        FinSi
        cont <- cont+1 //incremeto contador
    FinMientras

    Escribir "Cant. de numeros pares leidos: ", pares
    Escribir "Cant. de numeros impares leidos: ", impares
    Escribir "Suma de numeros pares: ", sumaParesares
    Escribir "Suma de numeros impares: ", sumaImpares
FinAlgoritmo
```

C

```c
//Algoritmo contarParesImapres
#include <stdio.h>
#include <conio.h>
int main (){
    int cantidad, cont, num, pares, impares, sumaPares, sumaImpares;
    cont = 1;
    pares = 0;
    impares = 0;
    sumaPares = 0;
    sumaImpares = 0;

    printf ("Ingrese cantidad de numeros \n");
    scanf ("%d", &cantidad);

    while (cont <= cantidad){

        printf("Ingrese numero \n");
        scanf("%d", &num);

        if (num % 2 == 0){
            pares++;
            sumaPares = sumaPares+num;
        }else{
            impares++;
            sumaImpares= sumaImpares+num;
        }

        cont++;

    }

    printf ("Cant. de numeros pares leidos: %d \n", pares);
    printf ("Cant. de numeros impares leidos: %d \n", impares);
    printf ("Suma de numeros pares: %d \n", sumaPares);
    printf ("Suma de numeros impares: %d", sumaImpares);
}
```

REPETIR PARA

Pseint

```
Algoritmo contarParesImpares
    Escribir "Ingrese cantidad de numeros"
    Leer cantidad

    //inicializo los contadores y acumuladores
    pares <- 0
    impares <- 0
    sumaPares <- 0
    sumaImpares <- 0

    Para cont<-1 Hasta cantidad Con Paso 1 Hacer
        Escribir "Ingrese numero"
        Leer num
        Si (num MOD 2 = 0) Entonces //si el numero leido es par
            pares <- pares+1 //incremento contador de pares
            sumaPares <- sumaPares+1 //acumulo los numeros pares
        SiNo
            //si el numero leido es impar
            impares <- impares+1 //incremento contador de impares
            sumaImpares <- sumaImpares+1 //acumulo los numeros impare
        FinSi
        cont <- cont+1 //incremeto contador
    FinPara

    Escribir "Cant. de numeros pares leidos: ", pares
    Escribir "Cant. de numeros impares leidos: ", impares
    Escribir "Suma de numeros pares: ", sumaParesares
    Escribir "Suma de numeros impares: ", sumaImpares
FinAlgoritmo
```

C

```c
//Algoritmo contarParesImapres

#include <stdio.h>
#include <conio.h>

int main (){
    int cantidad, cont, num, pares, impares, sumaPares, sumaImpare
    pares = 0;
    impares = 0;
    sumaPares = 0;
    sumaImpares = 0;

    printf ("Ingrese cantidad de numeros \n");
    scanf ("%d", &cantidad);

    for (cont = 1; cont<=cantidad; cont++){

        printf("Ingrese numero \n");
        scanf("%d", &num);

        if (num % 2 == 0){
            pares++;
            sumaPares = sumaPares+num;
         }else{
            impares++;
            sumaImpares= sumaImpares+num;
        }

    }

    printf ("Cant. de numeros pares leidos: %d \n", pares);
    printf ("Cant. de numeros impares leidos: %d \n", impares);
    printf ("Suma de numeros pares: %d \n", sumaPares);
    printf ("Suma de numeros impares: %d", sumaImpares);
}
```

REPETIR HASTA

Pseint

```
Algoritmo contarParesImpares
    Escribir "Ingrese cantidad de numeros"
    Leer cantidad

    cont <- 1
    pares <- 0
    impares <- 0
    sumaPares <- 0
    sumaImpares <- 0

    Repetir
        Escribir "Ingrese numero"
        Leer num
        Si (num MOD 2 = 0) Entonces
            pares = pares+1
            sumaPares <- sumaPares+num
        SiNo
            impares <- impares+1
            sumaImpares <- sumaImpares+num
        FinSi

        cont <- cont+1
    Hasta Que (cont > cantidad)

    Escribir "Cant. de numeros pares leidos: ", pares
    Escribir  "Cant. de numeros impares leidos: ", impares
    Escribir  "Suma de numeros pares: ", sumaPares
    Escribir "Suma de numeros impares: ", sumaImpares
FinAlgoritmo
```

C

```c
//Algoritmo contarParesImapres
#include <stdio.h>
#include <conio.h>
int main (){
    int cantidad, cont, num, pares, impares, sumaPares, sumaImpares;
    cont = 1;
    pares = 0;
    impares = 0;
    sumaPares = 0;
    sumaImpares = 0;

    printf ("Ingrese cantidad de numeros \n");
    scanf ("%d", &cantidad);

    do{

        printf("Ingrese numero \n");
        scanf("%d", &num);

        if (num % 2 == 0){
            pares++;
            sumaPares = sumaPares+num;
         }else{
            impares++;
            sumaImpares= sumaImpares+num;
        }

        cont++;

    }while(cont <= cantidad);

    printf ("Cant. de numeros pares leidos: %d \n", pares);
    printf ("Cant. de numeros impares leidos: %d \n", impares);
    printf ("Suma de numeros pares: %d \n", sumaPares);
    printf ("Suma de numeros impares: %d", sumaImpares);
}
```

CICLOS REPETITIVOS EN PSEINT Y LENGUAJE C

Ciclo repetitivo mientras en PSEINT

```
Algoritmo Sumar_Numeros_Aleatorios
    Sumar<-0//La variable Suma ACUMULA la suma de los numeros leidos
    Contador<-1 // La variable Contador CUENTA la cantidad del bucle
    Escribir "Ingrese la cantidad de numeros"
    Leer N
    Mientras (Contador <=N) Hacer //Mientras la Condicion sea Verdadera, se ingresa al bucle
        Escribir "Ingresar el numero ",Contador

        Leer num //Por cada vuelta se lee cada uno de los numeros

        Sumar<-Sumar + num // Suma el numero con la variable Sumar

        Contador<-Contador + 1 // Incrementa el Contador en 1. Pasa al siguente numero
    FinMientras // Termina el bucle Mientras cuando la condicion es Falsa

    Escribir "La Suma de los numeros es:", Sumar //En la Salida Escribe el valor de la suma

FinAlgoritmo
```

Ciclo repetitivo while en DEV C++

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

int main()
{
    int Contador,Suma,num,N; // Declaramos las variables de tipo entero
    Suma=0; //Inicializamos las variables Suma y Contador
    Contador=1;
    printf("ingrese la cantidad de numeros: ");
    scanf("%d",&N);
    while (Contador <= N)// Mientras el Contador sea menor igual a N (Cantidad de numeros ingresados)
     {
       printf("Ingrese el numero %d: ",Contador);
       scanf("%d",&num); //En cada vuelta del bucle leemos un numero
       Suma=Suma+num; //Sumamos los numeros
       Contador=Contador+1; //Incrementamos la variable Contador
     }

    printf("La suma es: %d",Suma); //Escribimos la Suma
getch();

}
```

Ciclo repetitivo para en PSEINT

```
Algoritmo Sumar_HASTA_QUE_SUMA_MAYOR_A_120
    Sumar<-0//La variable Suma ACUMULA la suma de los numeros leidos
    Repetir

        Escribir "Ingresar el numero "

        Leer num //Por cada vuelta se lee cada uno de los numeros

        Sumar<-Sumar + num // Suma el numero con la variable Sumar

    Hasta Que (Sumar > 120)
    //El bucle se repite hasta que la condicion sea Verdadera

    Escribir "La Suma de los numeros es:", Sumar //En la Salida Escribe el valor de la suma

FinAlgoritmo
```

Ciclo repetitivo do - while en DEV C++

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

int main()
{
    int Suma,num; // Declaramos las variables de tipo entero
    Suma=0; //Inicializamos las variables Suma y Contador

do //Todo lo que esta dentro de las llaves del do, se repite
   // mientras las condicion sea Verdadera
  {
   printf("Ingrese el numero ");
       scanf("%d",&num); //En cada vuelta del bucle leemos un numero
       Suma=Suma+num; //Sumamos los numeros
  }

 while(Suma<50); //Mientras la condicion sea Verdadera

    printf("La suma es: %d",Suma); //Escribimos la Suma
getch();

}
```

Ciclo repetitivo para en PSEINT	Ciclo repetitivo for en DEV C++

```
Algoritmo Sumar_Numeros_Aleatorios
    Sumar<-0//La variable Suma ACUMULA la suma de los numeros leidos

    Escribir "Ingrese la cantidad de numeros"
    Leer N
    Para Contador<-1 Hasta N Con Paso 1 Hacer //Mientras la Condicion sea Verdadera, se ingresa al bucle
        Escribir "Ingresar el numero ",Contador

        Leer num //Por cada vuelta se lee cada uno de los numeros

        Sumar<-Sumar + num // Suma el numero con la variable Sumar

    FinPara // Termina el bucle Mientras cuando la condicion es Falsa

    Escribir "La Suma de los numeros es:", Sumar //En la Salida Escribe el valor de la suma

FinAlgoritmo
```

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

int main()
{
    int Contador,Suma,num,N; // Declaramos las variables de tipo entero
    Suma=0; //Inicializamos las variables Suma y Contador
    printf("ingrese la cantidad de numeros: ");
    scanf("%d",&N);
   for(Contador=1;Contador<=N;Contador++)// Mientras el Contador sea menor igual a N (Cantidad de numeros ingresados)
    {
      printf("Ingrese el numero %d: ",Contador);
      scanf("%d",&num); //En cada vuelta del bucle leemos un numero
      Suma=Suma+num; //Sumamos los numeros
    }

    printf("La suma es: %d",Suma); //Escribimos la Suma
getch();

}
```