Introducción a la Programación

Programación desde cero

Fernando Jiménez Ávila

Col. Benito Juárez, C.P. 57000
Nezahualcóyotl, Estado de México
WhatsApp: 55 4886 3559

http://www.ccapmexico.com
http://www.ccapmexico.com/aulavirtual
ccapmexico@capmexico.com
http://www.facebook.com/ccapmx

Material Didáctico de Apoyo para el Curso Básico
Introducción a la Programación.

Impreso y hecho en Nezahualcóyotl, Estado de México, Abril 2024.

Contenido

Introducción

Al mismo tiempo que aparecieron las primeras computadoras, aparecieron los primeros problemas en su utilización, ya que su programación y uso eran sumamente laboriosos y complejo, por lo que una de las primeras actividades que se desarrollaron consistió en mejorar los mecanismos para acceder más fácilmente a la computadora, es decir, manejarla sin la necesidad de conocer su funcionamiento.

Dentro de estos mecanismos destacan de manera particular los lenguajes de programación, hecho, con lo cual, se ha vuelto relativamente fácil comunicarse con la computadora.

En este material didáctico se pretende dar a conocer cuáles son los elementos más importantes que intervienen en el desarrollo de programas, para obtener así un buen diseño del mismo y realizar una escritura correcta de los mismos.

Todas esas etapas influyen para el buen diseño y desarrollo de programas, éstos de acuerdo con las características y las necesidades que se tengan que satisfacer en cada caso.

Para esto, se deberá tener una combinación de todos y cada uno de los elementos que integran el análisis, diseño y desarrollo de programas, sólo así se alcanzará el objetivo principal de la programación, que es el de lograr la solución adecuada a cada problema, y programarla a fin de que la computadora la realice de manera adecuada.

Introducción a la Programación

La computadora

La palabra computadora es un término del inglés computer para indicar un sistema procesador de datos, lo cual determina con fidelidad su función de procesar, manipular y elaborar datos.

La computadora es una máquina compuesta de elementos físicos (hardware) de tipo electrónico, capaz de realizar una gran variedad de trabajos, con gran velocidad y precisión, y depende determinantemente de los elementos lógicos (software) para su funcionamiento.

Los elementos lógicos permiten el buen desempeño del equipo físico de la computadora. Así, de manera general, el software está representado por las aplicaciones, programas, datos, sistema operativo, etc.

Se puede considerar que la computadora es una máquina electrónica programable para procesar datos, constituida, como se ha mencionado, por elementos físicos (hardware) y lógicos (software), que determinan sus acciones y realizan funciones repetitivas o complicadas, con extraordinaria velocidad de acuerdo con una serie de instrucciones secuencialmente ordenadas.

De igual modo, se puede definir a la computadora como un rápido y exacto sistema de manipulación de símbolos electrónicos y datos, cuyo diseño y organización permite aceptar y manipular automáticamente datos, que, al procesarse, producirán resultados, todo bajo la dirección de programas almacenados internamente de instrucciones detalladas paso a paso.

A través de la computadora, se pueden automatizar funciones o actividades de diversos tipos, desde el ámbito educativo hasta complejos cálculos científicos.

La computadora surge como un satisfactor a la necesidad que tiene el ser humano de realizar complicadas operaciones matemáticas y de tener a la mano la información que necesita de una manera rápida. Gracias a la computadora, se ha podido reducir espacio y tiempo y se ha ido eliminando el trabajo rutinario implicando con esto una reducción considerable en costos en cualquier tipo de organización.

La computadora permite el tratamiento de la información de una manera muy rápida y eficaz para ayudar a resolver problemas, generalmente, en los siguientes ámbitos:

- Gestión empresarial.
- Aplicaciones industriales.
- Aplicaciones técnico – industriales.
- Médico.
- Militar.
- Mercadotecnia.
- Logística.
- Científica.
- Educativo.

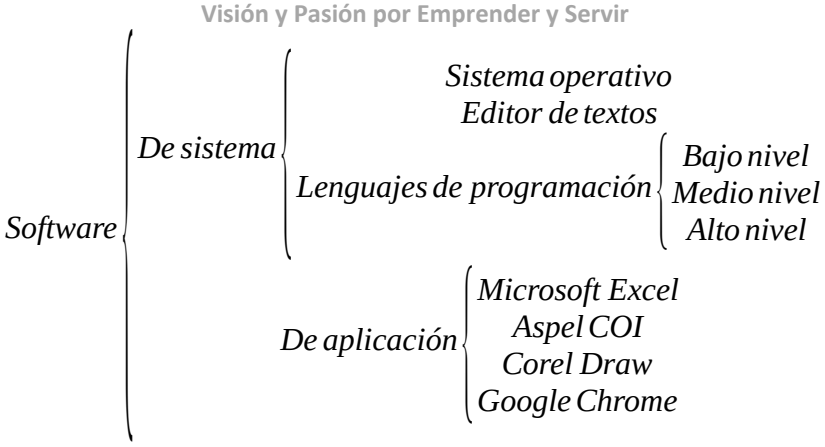
Componentes de la computadora

La computadora se divide en dos grandes componentes que son:

El hardware, que se refiere a la parte física de la computadora (circuitos de la computadora, dispositivos o periféricos de uso común, etc.).

- *Hardware*
  - *Dispositivos de entrada*
    - *Teclado*
    - *Ratón*
    - *Escáner*
    - *Lápiz óptico*
    - *Joystick*
    - *Lectora de tarjetas*
    - *CD ROM*
    - *DVD*
    - *Memoria USB*
    - *Lectora de código de barras*
  - *Dsipositivos de Procesamiento*
    - *CPU*
    - *Memoria RAM*
  - *Dispositivos de salida*
    - *Monitor*
    - *Impresora*
    - *Bocinas*
    - *CD ROM*
    - *DVD*
    - *Plóter*
    - *Memoria USB*
  - *dispositivos de almacenamiento*
    - *Disco duro*
    - *Disco flexible*
    - *Cinta magnética*
    - *CD ROM*
    - *DVD*
    - *Memoria ROM*
    - *Memoria USB*

El software, que se refiere a la parte lógica o a la programación de la computadora, desde un sistema operativo, los lenguajes de programación hasta las aplicaciones o paquetes de software.



Software de aplicación

Cuando se desarrolla un programa para controlar el proceso de una tarea determinada se le denomina aplicación. Muchos de estos programas son desarrollados para realizar trabajos únicos y específicos, de hecho, este tipo de software tiene un propósito específico, existe una gran cantidad de aplicaciones que se controlan a través de programas de carácter general. Esta clase de programas recibe el nombre de paquetes de software, por representar soluciones empaquetadas para el manejo de problemas de un mismo tipo.

En la actualidad, existe una gran variedad de paquetes de software, prácticamente para cualquier actividad desarrollada en las modernas empresas y diversidad de giros, la popularidad de este tipo de software se debe a la casi nula o mínima necesidad de conocer el funcionamiento de la computadora, y a la gran cantidad de documentación que proporcionan los fabricantes, proveedores y otros autores, así como las pantallas de ayuda en línea integradas como parte de la ejecución de los paquetes.

De esta forma, existen paquetes de software para aplicaciones tales como:

- Programas de contabilidad (Aspel COI, ContPaq).
- Programas de cálculo de nómina (Aspel NOI, NomiPaq).
- Procesadores de texto (Microsoft Word, LibreOffice Writer).
- Hojas de cálculo (Microsoft Excel, LibreOffice Calc).
- Manejadores o gestores de bases de datos (Microsoft Access, LibreOffice Base).
- Navegadores de Internet (Microsoft Edge, Google Chrome, Mozilla FireFox).

Software de sistema

Es el software de creación, manipulación, modificación y ejecución de programas o de software de aplicación. Lo integran el sistema operativo, los editores de texto y los lenguajes de programación.

El sistema operativo de una computadora es el conjunto de programas fundamentales para su funcionamiento y forma parte de lo que se denomina software de sistema o software de base, tiene como objetivo controlar óptimamente las partes constituyentes de la computadora, es decir, administrar los recursos, tales como la memoria, el control de los dispositivos de hardware, el procesador y, los más importante, la información, además de los procesos para el óptimo funcionamiento de la computadora.

Aunque el sistema operativo ofrece mucha asistencia al usuario para establecer comunicación con la computadora, existen programas dedicados a ciertas funciones específicas que facilitan esta tarea aún más.

Los editores de texto son programas de sistema que permiten al usuario crear, revisar y almacenar tanto archivos de texto en general como programas escritos en algún lenguaje de programación específico, o bien, textos de carácter general.

Los lenguajes de programación son programas que permiten escribir o desarrollar aplicaciones que sean necesarias. Los lenguajes de programación pueden clasificarse desde el punto de vista de sus aplicaciones, es decir, cómo se usan:

Bajo nivel (lenguaje máquina)

Los lenguajes máquina son aquéllos que están escritos en lenguajes directamente ejecutables por la computadora, ya que sus instrucciones son cadenas binarias, que especifican una operación y las posiciones de memoria implicadas en la operación y se denominan instrucciones de máquina o código máquina.

Para programar se usa el código máquina, es decir, el código binario, lo cual implica que la programación sea un tanto difícil.

Una instrucción en este lenguaje sería:

Dirección de memoria	Contenido
0100	0010 0010 0110 0001 1100 0101 0001 0101 1110
0101	0100 0001 0111 0101 1010 0101 1110 1010 0001
0111	0011 1000 0101 0110 0100 1000 0111 1000 0110

Las instrucciones en lenguaje máquina dependen del hardware, y, por lo tanto, diferirán de una computadora a otra. Lo cual imposibilita que un programa hecho en una arquitectura de hardware no funcionará en otra arquitectura de hardware, se dice entonces, que el programa no es transportable. El lenguaje de máquina es específico de la arquitectura de hardware, aunque el conjunto de instrcciones pueda ser similar entre arquitecturas de hardware distintas,

Si se quisiera mostrar el mensaje “Hola mundo” en código binario, la instrucción podría ser como sigue:

0011 1100 1010 1001 1001 0110 1010 0010 0101 1110 1010 0100 0101 **0100 1000 0110 1111 0110 1100 0110 0001 0010 0000 0110 1101 0111 0101 0110 1110 0110 0100 0110 1111** 0101 1010 1010 1010 0101 0101 0000 0011 0010 1010 0000 1111 0101 1010 0110 1001

Destacándose que lo que está en negritas es el mensaje, el resto son instrucciones necesarias para mostrar el mensaje. También se podría codificar como sigue:

48 6F 6C 61 2C 20 65 73 74 65 20 65 73-20 75 6E 20 20 72 6F 67 72 61 6D 61 20 68 65 63 68 6F 20 65 6E 20 61 73 73 65 6D 62 6C 65 72 20 70 61 72 61 20 6C 20 6C 61 20 57 69 6B 69 70 65 64 69 61 24

Lo anterior podría ser el equivalente del código binario en código hexadecimal, que, de igual forma, sigue siendo lenguaje máquina.

La ventaja de programar en lenguaje máquina es la posibilidad de cargar (transferir un programa directamente a la memoria) sin necesidad de un proceso posterior, lo que supone una velocidad de ejecución superior a la de cualquier otro tipo de lenguaje.

Los inconvenientes, en la actualidad, superan a las ventajas, lo que hace prácticamente no recomendables a los lenguajes de bajo nivel.

Los principales inconvenientes son:

- Dificultad y lentitud en la codificación.
- Poca fiabilidad.
- Dificultad de verificar y poner a punto los programas.
- Los programas sólo son ejecutables en el mismo procesador o arquitectura de hardware.

A final de cuentas, sin importar que un programa se escriba en un lenguaje de programación de medio nivel o de alto nivel, dichas instrucciones se traducen a código máquina. Es decir, las computadoras sólo entienden el código binario, en donde la secuencia de 0 (ceros) y 1 (unos) se convierten en acciones a realizarse.

Medio nivel (Lenguaje ensamblador)

Los lenguajes ensambladores son más fáciles de utilizar que los lenguajes máquina, pero al igual que ellos, dependen del hardware en particular. Las instrucciones en ensamblador son instrucciones conocidas como mnemotécnicos; por ejemplo, mnemotécnicos típicos de operaciones aritméticas son, por ejemplo: en inglés, ADD, SUB, DIV, etc.; en español serían SUM, RES, DIV, etc.

A modo de ejemplo, una instrucción en lenguaje ensamblador para sumar dos números, sería:

ADD, M, N, P

Esta instrucción significaría: sumar el número almacenado o contenido en la posición de memoria M al número almacenado en la posición de memoria N y situar el resultado de la suma en la posición de memoria P.

Evidentemente, es mucho más sencillo recordar la instrucción anterior con un mnemotécnico que con su equivalente en código binario.

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la computadora, en esto se diferencia esencialmente del lenguaje máquina, sino que requiere de un proceso de traducción al lenguaje máquina. El traductor de programas es un programa llamado ensamblador.

Los lenguajes ensambladores presentan la ventaja frente a los lenguajes máquina de su mayor facilidad de codificación y, en general, su velocidad de cálculo, al no realizar cálculos de traducción a código binario.

Los inconvenientes más notables son:

- Dependencia total del hardware, lo que impide la transportabilidad de los programas (es decir, la posibilidad de ejecutar un programa en diferentes máquinas).
- La formación de los programadores es más compleja, ya que exige no sólo el conocimiento de las técnicas de programación, sino también el conocimiento del interior o funcionamiento de la computadora o arquitectura de hardware.

Actualmente, los lenguajes ensambladores tienen sus aplicaciones muy reducidas en la programación de aplicaciones y se centran en aplicaciones de tiempo real, contra procesos y de dispositivos electrónicos, entre otros usos.

Si se quisiera mostrar el mensaje “Hola mundo”, el código en lenguaje ensamblador podría ser:

```
1.  .model small
2.  .stack
3.  .data
4.      saludo   db "Hola mundo", "$"
5.  .code
6.  main  proc                  ;Inicia proceso
7.      mov   ax,seg saludo     ;hmm ¿seg?
8.      mov   ds,ax             ;ds = ax = saludo
9.      mov   ah,09             ;Function(print string)
10.     lea   dx,saludo         ;DX = String terminated by "$"
11.     int   21h               ;Interruptions DOS Functions
12.                             ;mensaje en pantalla
13.     mov   ax,4c00h          ;Function (Quit with exit code (EXIT))
14.     int   21h               ;Interruption DOS Functions
15. main  endp                  ;Termina proceso
16. end main
```

Otro ejemplo en ensamblador:

```
;-------------------------------------------
; Programa que imprime un string en la pantalla
;-------------------------------------------
        .model small                        ; modelo de memoria
        .stack                              ; segmento del stack
        .data                               ; segmento de datos
        Cadena1 DB 'Hola Mundo.$            ; string a imprimir (finalizado en $)
        .code                               ; segmento del código
;-------------------------------------------
; Inicio del programa
;-------------------------------------------
        programa:
                ;---------------------------------------
                ; inicia el segmento de datos
                ;---------------------------------------
                MOV AX, @data               ; carga en AX la dirección del segmento de datos
                MOV DS, AX                  ; mueve la dirección al registro de segmento por medio de AX
                ;---------------------------------------
                ; Imprime un string en pantalla
                ;---------------------------------------
```

```
        MOV DX, offset Cadena1 ; mueve a DX la dirección del string a imprimir
        MOV AH, 9              ; AH = código para indicar al MS DOS que imprima en la pantalla, el string en DS:DX
        INT 21h                ; llamada al MS DOS para ejecutar la función (en este caso especificada en AH)
        ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        ; Finaliza el programa
        ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        INT 20h                ; llamada al MS DOS para finalizar el programa
end programa
```

Como se puede observar en los ejemplos anteriores, el lenguaje ensamblador implementa una representación simbólica del código binario para programar en una arquitectura de hardware; cada arquitectura de hardware tiene su propio lenguaje ensamblador, que es definida por el fabricante del hardware.

El lenguaje ensamblador está basado en los mnemotécnicos que representan las instrucciones, los registros del procesador, las posiciones de memoria y otras características del lenguaje.

Alto nivel (de propósito general, específicos, científicos, etc.)

Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que los programadores escriban y entiendan los programas de un modo más fácil que los lenguajes de bajo nivel y de medio nivel. Otra razón es que un programa escrito en un lenguaje de alto nivel es prácticamente independiente de la computadora; esto es, las instrucciones del programa no dependen del diseño del hardware o de una computadora en particular.

En consecuencia, los programas escritos en lenguajes de alto nivel, son prácticamente transportables, lo que significa la posibilidad de poder ser ejecutados con pocas o ninguna modificación en diferentes tipos de computadoras (siempre que sean compatibles a nivel del procesador); al contrario que los programas escritos en lenguaje máquina o en lenguaje ensamblador que sólo se pueden ejecutar en un determinado tipo de computadora.

Los lenguajes de alto nivel presentan las siguientes ventajas:

- El tiempo de formación de los programadores es relativamente corto en comparación de los dos nivele anteriores.
- La escritura de los programas se basa en reglas sintácticas similares a los lenguajes humanos, por lo regular, en inglés, por ejemplo: read, write, input, print, scanf, printf, cint, cout, if, else, select, case, switch, repeat, for, while, etc.
- Las modificaciones y puestas a punto de los programas son más fáciles.
- Reducción del costo de los programas.
- Transportabilidad.

Los inconvenientes se concretan en:

- Incremento del tiempo de puesta a punto, al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo.
- No se aprovechan los recursos internos de la computadora, que se explotan al máximo y mucho mejor en lenguajes máquina y ensamblador.
- Aumento de la ocupación de la memoria.
- El tiempo de ejecución de los programas es mucho mayor.

Al igual que sucede con los lenguajes ensambladores, los programas fuentes tienen que ser traducidos por programas traductores, en este caso, por programas compiladores o por programas intérpretes.

Para mostrar el mensaje “Hola mundo” en un lenguaje de programación de alto nivel, los ejemplos podrían ser:

Ada

```
with Ada.Text_IO;

procedure HolaMundo is

begin

  Ada.Text_IO.Put_Line (“Hola mundo”);
```

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HOLA-MUNDO.
PROCEDURE DIVISION.
   DISPLAY “Hola mundo”.
   STOP RUN.
```

Pascal

```
Program HolaMundo;

Begin

   WriteLn('Hola mundo');

End.
```

end HolaMundo;

BASIC

PRINT “Hola mundo”

Forth

." Hola mundo" CR

Prolog

main() :- write("Hola mundo"), nl.

C

```
#include <stdio.h>

int main()

{
   printf("Hola mundo");

  return 0;
}
```

Fortran

```
program Hola

  print *, "Hola Mundo"

end program HolaMundo
```

Rust

```
fn main() {

    println!("Hola mundo");

}
```

C++

```
#include <iostream>
using namespace std;

int main()

{
  cout << "Hola mundo" << endl;

  return 0;
}
```

Java

```
public class Main {

    public static void main(String[] args) {

        System.out.println("Hola Mundo");

    }
}
```

Python

```
print("Hola mundo")
```

Ruby

```
puts "Hola mundo"
```

Dart

```
print("Hola mundo");
```

Swift

```
print("Hola mundo")
```

C#

```
public class HolaMundo
   {
      public static void Main()
      {
            System.Console.WriteLine("Hola
mundo");
   }
}
```

o usando instrucciones de nivel superior (a partir de C#v9):

```
Console.WriteLine("Hola mundo");
```

JavaScript

Para consola de navegador/Javascript runtime (Como por ejemplo Node.js):

```
console.log("Hola mundo");
```

para un documento HTML:

```
document.write("Hola mundo");
```

o

```
alert("Hola mundo");
```

o como función en JS:

```
export function HolaMUndo()

{return "Hola mundo";}
```

TypeScript

```
process.stdin.resume();
process.stdin.setEncoding('utf8');

console.log("Hola mundo")
```

Visual Basic

PHP

Dbase III+

```
public class compiler

 shared function Main as integer

   Console.WriteLine ("Hola mundo")

   return 0

 End function

end class
```

```
<?php

echo 'Hola mundo';

?>
```

```
CLEAR
@0, 2 SAY 'Hola mundo'
RETURN
```

Go

```
package main
import "fmt"

func main(){
      fmt.Println("Hola mundo")
}
```

Kotlin

```
fun main(args: Array<String>) {

   println("Hola mundo")

}
```

Clojure

```
(println "Hola mundo")
```

Objetive-C

```
#import <Foundation/Foundation.h>
int main(void){

            NSAutoreleasePool*  pool  =
[[NSAutoreleasePool alloc] init];

   NSString *s = @"Hola mundo";
                              [[NSFileHandle
fileHandleWithStandardOutput] writeData:
                  [s      dataUsingEncoding:
NSUTF8StringEncoding]];
   [pool release];

 return 0;
}
```

MySql

```
create table HolaMundot(id integer, title varchar(100));
insert into HolaMundo (id, title) values(1, "Hola mundo");
select * from HolaMundo;
```

Haskell

```
main = putStrLn "Hola mundo"
```

Qué es programar

Programar es proporcionar a la computadora instrucciones, a modo de programa, las cuales deben ser específicas y directas, indicando claramente el orden de realización de cada parte o instrucción e indicar cuándo dejar de hacer algo y continuar con otra tarea.

Programa

Es un conjunto de acciones, en este caso, de instrucciones, que van encaminadas a un fin específico para realizar funciones o actividades que han de resolver problemas mediante una computadora. Dichas instrucciones deben ser especificadas por el programador.

Programador

Un programador o una programadora es quien elabora, desarrolla, escribe o codifica programas de computadora, creando de esta forma, software, por medio de un lenguaje de programación. A los programadores también se les denomina desarrolladores de software.

Características del programador

- ✓ Ser capaz de manejar un problema hasta resolverlo a la perfección.
- ✓ Saber enumerar una serie de pasos por medio de los cuales se va a llegar a la solución del problema, tomando en consideración las capacidades de la computadora.

- ✓ Tener capacidad de análisis y lógica.
- ✓ Conocer el lenguaje de programación adecuado a la solución del problema.
- ✓ Ser autodidacta.
- ✓ Ser realista y ser consciente de las limitaciones.
- ✓ Flexibilidad y adaptación al cambio.

El programador debe tener todas las características enumeradas anteriormente, además, debe realizar lo siguiente:

- ➢ Dado el problema, debe encontrar un método para resolverlo (algoritmo a emplear).
- ➢ El método debe ser transcrito de tal forma que la computadora pueda entenderlo (codificar el programa).
- ➢ Ejecutar el programa y verificar su correcto funcionamiento.
- ➢ Documentar el programa correcto para facilitar su posterior utilización.

No hay que olvidar que la práctica es fundamental para adquirir y afinar las características antes mencionadas.

Conceptos del procesamiento de información

La información representa una forma de comunicación de datos, textos, imágenes o audio (multimedia), y es considerada como el resultado final del procesamiento de datos, hechos y fenómenos en datos organizados con significado y utilidad.

Entrada

La entrada de datos consiste en suministrar los datos requeridos o que necesita el programa, con el objeto de que, a través del procesamiento, se genere la información esperada.

El proceso de entrada requiere de un mecanismo o medio (dispositivo de entrada) que lo permita, por ejemplo: el teclado, lectora de caracteres, lectora de tarjetas, escáner, lápiz óptico, etc.

Salida

La salida de información representa el objetivo final del procesamiento de datos. El proceso de salida registra la información generada por la computadora (a través de la entrada de datos y su manipulación), presentada en pantalla (a modo de consulta) o en impresa en papel (a modo de reporte).

Edición

Editar es cambiar el valor o contenido de un dato, ya sea una constante o una variable, capturado o introducido a la computadora previamente.

Manipulación

Para poder manipular la información que se necesite, es necesario contar con el equipo adecuado para obtener la capacidad de recuperación de los datos, rápida y económicamente, del almacenamiento de los datos, procesar los datos a mayor velocidad y con menores costos; facilitar la entrada de los datos, recuperarlos, procesarlos y presentarlos como información útil.

Básicamente, todo lo anterior, es llevado a cabo por la Unidad Central de Proceso, que controla todas las operaciones de la computadora.

Almacenar

Antes de cualquier tratamiento aplicado a los datos, éstos deben estar, cuando menos, temporalmente almacenados a través de algún medio. En el contexto de la computación, el almacenamiento de los datos se hace a través de la memoria.

La unidad de memoria es en donde se almacenan las instrucciones y los datos necesarios para la ejecución de un determinado proceso. La memoria de almacenamiento interno se utiliza para conservar las instrucciones (programas) que se procesarán, los datos de entrada a los que se les aplicará un proceso o una salida y los datos de referencia asociados con el procesamiento.

Recuperar

La recuperación de datos o de información consiste en obtener los datos, almacenados previamente, de la manera más rápida y con menores costos, en la memoria o en dispositivos externos (cintas, discos, memorias USB, etc.), que el programa necesita para procesar los datos y obtener la información deseada.

Asignar

Asignar es dar un contenido o un valor a una constante o variable, que se puede o no modificar durante el procesamiento del programa. Por ejemplo:

a = 2 IVA = 0.16 variable = argumento variable = expresión

Comparar

Comparar es examinar semejanzas o igualdades y diferencias que haya o existan entre los datos a procesar, y en base a esa comparación, tomar una decisión o realizar alguna acción.

Obtener

Obtener es convertir tipos numéricos a alfanuméricos y viceversa; convertir letras mayúsculas a minúsculas y viceversa, obtener el número de caracteres (longitud) de una cadena de caracteres, entre otras operaciones necesarias para el procesamiento de los datos

Destruir

La fase final del ciclo de vida de los datos es su destrucción o eliminación; sin embargo, éstos se pueden volver a almacenar para su uso posterior. Los datos pueden eliminarse directamente después de su uso, mediante una revisión, esto está en función de su importancia para borrarlos.

Metodología para la Solución de Problemas

Desarrollo de un programa

La programación requiere de algo más que la simple instrucción de las órdenes y de sentencias escritas correctamente, de hecho, es la parte más sencilla. La programación requiere de una cuidadosa planificación del código, y después de una rigurosa prueba.

Desafortunadamente, no hay una única forma correcta de escribir un programa ni determinar cuándo es bueno o eficiente. Sin embargo, la mayoría de los programadores tienen una tendencia a seguir de nueve pasos en el diseño y desarrollo de un programa, los cuales son:

1. Definir el problema.
2. Planear o bosquejar la solución.
3. Realizar, representar y seleccionar el algoritmo adecuado.
4. Realizar el diagrama de flujo.
5. Codificar.
6. Verificar y depurar el programa.
7. Probar y validar el programa.
8. Documentar los pasos 1 al 5.
9. Dar mantenimiento al programa.

Pasos de la metodología para la solución de problemas

Como se ha mencionado, para poder realizar un programa, es necesario seguir con una serie de pasos que ayuden a diseñar y desarrollar la solución de una manera adecuada.

Definición del problema

La definición del problema se centra en la elaboración de un texto que permitirá saber qué es lo que se pretende solucionar, y permitirá proporcionar información útil para el planteamiento de la solución.

La creación de un programa efectivo comienza con un buen diseño y una visión amplia del contenido y propósito del programa. La forma en que se organice un programa dependerá de la información que contenga el programa, de la comprensión que se tenga del mismo y de la manera en que se espera que sea utilizado el programa.

No hay que olvidar que se crean programas para resolver problemas, para que sean utilizados, y no simplemente para que sean observados.

Dado que las decisiones que se tomen en un principio afectarán a todo el programa, se debe pensar cuidadosamente en cada detalle del problema, para determinar una solución adecuada.

Este paso, a menudo es olvidado, incluso por programadores profesionales, debido a las prisas por crear el programa, el problema puede ser tan simple como el querer automatizar una sencilla tarea, pero en el proceso de la definición del problema se tiene que determinar qué es lo que se espera que haga el programa.

Planeación o bosquejo de la solución

En este paso se debe entender por completo el problema, y dejar en claro cuál es la información útil que se da, qué otra información hace falta y con base en esto, determinar la mejor solución.

La entrada de datos, es la forma en que se puede usar el programa para situar los datos, ha de ser así mismo considerada durante el proceso de la definición del problema.

Para la forma en que serán introducidos los datos que usará el programa, se debe disponer de un método lógico para poner la información en la computadora.

La salida de los datos es la información que el programa debe producir, tiene que ser detenidamente considerada. La salida es muy útil en forma de informe impreso, así que, definir qué tipo de salida se necesita, a menudo es similar al proceso de definir qué se necesita en un informa, destacándose que lo mínimo de información necesaria es lo mejor.

En general, durante este paso se define lo siguiente:

- Entrada.
- Salida.
- Datos adicionales.
- Solución.

Un buen diseño hace uso de control del programa para realizar los procesos de entrada y salida de información de forma clara y provechosa.

Realización, representación y selección del algoritmo adecuado

El algoritmo es una serie de datos ordenados, encaminaos a resolver el problema en cuestión. Es un conjunto finito de instrucciones precisas que realizan una tarea, una actividad o un proceso, el cual determinará un resultado.

En términos simples, se describe como la manera de cómo el programa ha de resolver un problema como conjunto de instrucciones que indican con precisión cómo obtener los resultados deseados.

Un algoritmo es una secuencia ordenada de operaciones bien definidas, que la ejecutarse producirán siempre un mismo resultado, y se terminará en un espacio de tiempo definido. En un algoritmo, cada operación debe estar bien definida, claramente comprensible y sin ambigüedades para la persona o computadora que la ejecute.

Para poder realizar un algoritmo, se debe tener definido lo siguiente:

1. Pedir datos.
2. Desplegar datos.
3. Evaluar condiciones.
4. Ejecutar operaciones.
5. Desplegar resultados.

Así, las principales características que debe tener un algoritmo son:

a) Describir los pasos a seguir y el orden de los mismos.
b) Ser definido (no presentar ambigüedades).
c) Ser general (debe procesar diversos juegos de datos).
d) Ser finito en tamaño y tiempo.

Diagrama de flujo

Un diagrama de flujo es la forma gráfica de representar un algoritmo, el cual se compone de un plano o un diagrama lógico de la solución del problema.

A la representación gráfica de un algoritmo se le llama diagrama de flujo; para llegar a la solución de un problema es necesario definirlo antes y diseñar un esquema de la solución. La solución requiere de tres elementos: datos del problema, resultados esperados y el proceso de solución (algoritmo).

La forma de presentar los algoritmos varía de acuerdo con la conveniencia de cada programador; sin embargo, la utilización de diagramas de flujo para obtener una representación gráfica de la solución del problema, ofrece una herramienta útil y con muchas ventajas.

Existen varias maneras de realizar dichos diagramas de flujo. Para realizar un diagrama de flujo se utilizan símbolos o cajas estándar, y tienen los pasos del algoritmo escritos en esas cajas unidas con flechas denominadas líneas de flujo, que indican la secuencia en que se deben ejecutar dichos pasos, los símbolos usados son:

Para construir algoritmos eficientes y funcionales, es conveniente seguir los siguientes pasos:

1. Los diagramas de flujo deben tener un principio y un final, ya que podrían ser utilizados como submódulos de un módulo superior.
2. Las flechas o líneas de conexión, en la media de lo posible, deben ser rectas y sólo verticales u horizontales. Se puede recurrir a conectores debidamente enumerados, pero sólo en casos muy necesarios.
3. Ambos extremos de las flechas deben estar conectados con los símbolos permitidos, otras flechas, conectores, etc.
4. Los símbolos deben estar dibujados de tal forma que representen el proceso de arriba hacia abajo y de izquierda a derecha.
5. Se deben realizar diagramas centrados con respeto a la hoja donde se construyan.
6. Evitar la utilización de términos específicos de algún lenguaje de programación.
7. Hacer comentarios al margen, para lo cual, se utiliza el símbolo correspondiente, respecto a las variables utilizadas en él.
8. Preferentemente, en las operaciones lógicas debe usarse

   si x = y

   Y no hacerlas complicadas o difíciles de entender

   si no es x <> y

9. Por lo regular, un símbolo o bloque, se accede por arriba o por la izquierda, y se continúa por abajo o por la derecha.
10. Es conveniente que un diagrama d flujo se represente en una sola página.

Codificación

Es donde la solución del problema se escribe en algún lenguaje de programación, el cual puede ser de bajo nivel, medio nivel o alto nivel, dependiendo de las características del problema a solucionar. Llega el momento de escribir propiamente el programa.

Un programa es un listado de acciones o instrucciones que realizarán una actividad o un proceso con la ayuda de una computadora.

Un programa de computadora es un conjunto de instrucciones (órdenes dadas a la computadora), que producirán la ejecución de una determinada tarea. En esencia, un programa es un medio para conseguir un fin.

El fin, obviamente, es la información que debe producir el programa al procesar los datos necesarios para solucionar el problema.

Para desarrollar un programa, se requiere realizar una serie de pasos, entre los que se incluyen los siguientes:

- ✓ Anotar la fecha.
- ✓ Anotar nombre de quien realice el programa.
- ✓ Anotar nombre del archivo o programa.
- ✓ Anotar el número de versión, revisión o actualización.
- ✓ Anotar lo objetivos del programa, determinando los requisitos básicos que se deben cumplir.
- ✓ Escribir la solución con las instrucciones del lenguaje de programación.
- ✓ Especificar las estructuras de datos de entrada y salida.
- ✓ Especificar rangos aceptables de variables de entrada.
- ✓ Especificar mensajes de error.
- ✓ En caso de ser módulos de programa, especificar qué programa llama a otro.
- ✓ Incluir comentarios.
- ✓ Numerar la hojas o páginas de la siguiente forma: 1 de 3, 2, de 3, 3 de 3, etc.

La escritura de buenos comentarios es muy importante en los programas. Un buen comentario facilita la legibilidad del programa, naturalmente, no influye en la codificación; sin embargo, un mal comentario puede deteriorar la codificación.

Los comentarios deben apoyar la codificación, y aunque no existen sitios fijos dónde situarlos, se deben considerar deseables los siguientes puntos:

- ➢ Declaraciones de variables.
- ➢ Estructuras de control.
- ➢ Llamadas y definiciones de subprogramas, subrutinas o funciones.
- ➢ Codificaciones difíciles o complejas.

Verificación y depuración

En este paso, se revisa exhaustivamente la codificación del programa hasta que no tenga errores. Cualquier error del programa se corrige mediante este paso

También se examina el programa para ver si realmente se han cubierto todas las necesidades, si no, se tienen que hacer los cambios necesarios o añadir lo que haga falta y hacer cualquier mejoría que se necesite.

La verificación tiene la intención, como en la prueba, de hallar errores, se lleva a cabo ejecutando el programa en un ambiente simulado, es decir, con datos de prueba.

Prueba y validación

En este paso, se utiliza el programa para verificar la operatividad del mismo. Y así, de esta forma comprobar si el programa funciona como se diseñó, es decir, si realiza todo lo que se espera que haga el programa.

La prueba se efectúa para demostrar que no hay errores en un programa. Sin embargo, debe tenerse en cuenta que esto es prácticamente imposible, puesto que ningún programa puede demostrar que esté limpio de errores.

De hecho, este paso se realiza con la intención explícita de hallar errores, es decir, hacer que le programa falle, y así, hacer las correcciones necesarias.

La validación se refiere al proceso del uso del software o programa en un ambiente no simulado, en una situación real, para hallar errores, es decir, con datos reales.

A los dos pasos anteriores (verificación y depuración, prueba y validación) se les conoce también como puesta a punto de un programa, y como se ha expuesto, se realiza lo siguiente:

- Detección de errores.
- Depuración de errores.
- Prueba del programa.

Documentación

En este paso es donde se deben almacenar los pasos del 1 al 4, junto con el listado de la codificación de la solución del problema. La documentación de un programa toma una de las formas de instrucciones escritas, como un manual, que explica cómo funciona el código, y comentarios en el programa, mismo sobre cómo se ha diseñado el mismo.

Los comentarios son simplemente unja ayuda para que se modifique el programa. No sólo hace más fácil de entender el programa, sino que, si otra persona debe realizar cambios, la tarea será mucho más fácil.

La documentación interna incluye:

- Comentarios.
- Codificación.
- Listado (diagramas de flujo, algoritmos, declaraciones de variables, etc.).

El objetivo de la documentación interna es hacer al programa legible, comprensible y fácil de modificar.

La documentación externa incluye:

- Manual de usuario.
- Manual del operador.
- Manual de mantenimiento del programa.
- Lista de datos de prueba (tests) y resultados.
- Listado de errores.
- Diseño descendente con detalle de programas y subprogramas
- Versiones modificadas en uno y diferencias entre sí.

La importancia de la documentación externa radica en lo siguiente: el programa ha sido escrito por un programador y debe ser consultado por distintas personas, y los programas pueden contener errores, pese a que el programa en apariencia funciona, y deberán ser verificados por personas distintas al programador.

En resumen, la documentación incluye:

a) Elaborar una bitácora o una carpeta de descripción de todo lo que se realizó durante el análisis, diseño y desarrollo del programa.
b) Copias del programa.
c) Elaboración del manual.

Mantenimiento del programa

El mantenimiento debe ser un aspecto más del diseño de los programas. Efectuar cambios y ajustes, no necesariamente indican la corrección de errores o la ocurrencia de fallas o problemas.

Entre los cambios más frecuentes se encuentra la adición de información. Se pueden revisar los requerimientos del programa como consecuencia de su uso o del cambio en las necesidades de operación. Quizás sea necesario corregir algún descuido que ocurrió durante el diseño.

A menudo, surge la necesidad de capturar más datos y procesarlos. Quizás sea necesario añadir características para la detección de errores con la finalidad de evitar que quienes utilicen el programa emprendan por equivocación una acción no deseada.

Los cambios en las condiciones de operación en las necesidades de los usuarios, hacen necesario el mantenimiento o la modificación continua de los programas y sistemas existentes.

Todas estas situaciones son realidades del mantenimiento de programas.

Cuando se presentan, son un buen indicador de que el programa se está utilizando, de que tiene una función útil.

Lógica Computacional

Dato

En sentido estricto, dato representa la unidad de información la cual puede definirse con precisión, es decir, los datos se definen como elementos que denotan valores, magnitudes, estados, etc., por ejemplo: nombre, apellido paterno, apellido materno, edad, sexo, dirección, número de teléfono, número de celular, CURP, RFC, clave de INE, número de nómina, número de seguridad social, salario, etc.

Tipos de datos

La computadora puede manejar básicamente cuatro tipos de datos, los cuales son:

Datos numéricos

Son datos cuyo contenido es una serie de dígitos o números (del 0 al 9), que en conjunto proporcionan un valor numérico. La computadora puede manejar dos tipos: el tipo entero y el tipo real

El tipo entero es un subconjunto infinito de los números reales. Los enteros son números completos, no tienen componentes fraccionarios o decimales y están formados por los números positivos, el cero y los números negativos.

A estos números, se les denomina en ocasiones, como números de punto o coma fija.

Los tipos de datos enteros que se pueden encontrar en los lenguajes de programación, son:

integer, int, long, longint, byte, bitset, word, shortint, cardinal

El tipo real consiste en el conjunto de los números reales. Los números reales siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consta de una parte entera y de una parte decimal.

En aplicaciones científicas se requiere de una representación especial para manejar números o muy grandes o muy pequeños. Existe un tipo de representación denominado notación exponencial o notación científica, y que se utiliza para dichos números.

A los números reales también se les conoce como números de coma o punto flotante, y es una generalización de la notación científica, en estas expresiones, se considera la mantisa (parte decimal) como número real y al exponente (parte potencial o exponencial) como potencia de base diez.

Por ejemplo:

$$3.3456E2 = 3.3456 \text{ X } 10^2$$

Los tipos de datos reales que se pueden encontrar en los lenguajes de programación, son:

real, float, currency, single, double

Existen algunos lenguajes de programación, como FORTRAN, que admiten otros tipos de datos; por ejemplo, para manejo de números complejos (complex), que permite tratar, precisamente, con los números complejos; y otros lenguajes, que si bien, no disponen de algunos tipos de datos, por ejemplo, para manejar números complejos, peros sí permiten declarar y definir tipos de datos propios.

Datos alfanuméricos

Son datos cuyo contenido son letras, números, caracteres especiales, o bien, una combinación de ellos.

El tipo alfanumérico consiste en una cadena de caracteres o de un solo carácter, una cadena es una secuencia de cero, uno, dos o más caracteres que incluye letras del alfabeto, dígitos, caracteres especiales y espacios en blanco.

Los tipos de datos alfanuméricos que se pueden encontrar en los lenguajes de programación, son:

string, char

Datos lógicos

Son datos que sólo pueden tomar uno de dos valores: verdadero o falso. Este tipo de datos, también denominado booleano, se utiliza para representar las alternativas sí/no a determinadas condiciones. Por ejemplo, cuando se pide saber si un valor entero es par, será verdadero o falso, según sea par o impar.

En este tipo de datos, sólo existen dos constantes lógicas: verdadero (true) y falso (false) y el resultado también es lógico. De igual forma, se puede utilizar el número 1 para verdadero y el número 0 para falso.

Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas utilizando los operadores lógicos: no (not), y (and), o (or), imp, eqv, xor.

En las expresiones lógicas se pueden combinar operadores de relación y lógicos.

Los operadores de relación se pueden aplicar a cualquiera de los tipos de datos: entero, real, cadena de caracteres, carácter.

Los tipos de datos lógicos o booleanos que se pueden encontrar en los lenguajes de programación, son:

logical, boolean

Zona de memoria

Una zona de memoria está formada por un nombre y un contenido, es decir, un nombre que lo identifica y un contenido para los valores que almacenará, ya sea l valor de una constante o el valor de una variable.

Constantes

Una constante contiene un valor que no cambiará durante la ejecución del programa al procesar la información.

Variables

Una variable contiene un valor que puede modificarse a lo largo de la ejecución de un programa.

Tanto las constantes como las variables tienen atributos propios como:

Nombre: es el que se va a utilizar para referirse a la constante o a la variable en el programa.
Tipo: el tipo determina qué clase de valor puede almacenar la constante o la variable.
Ámbito: el ámbito de una constante o de una variable especifica en qué parte del programa son conocidas y, por lo tanto, en dónde se pueden utilizar, que puede ser local o global.

Reglas para nombrar constantes y variables

Los nombres de estas zonas de memoria se forman bajo las siguientes reglas:

1. El primer carácter debe ser una letra.
2. Los demás caracteres podrán ser letras, dígitos o caracteres especiales.
3. La longitud debe tener como máximo 64 caracteres, aunque se prefiere no rebasar de 32 caracteres.
4. El nombre que se le asigne a la zona de memoria, no debe ser igual al de algún comando, función, procedimiento, subrutina o palabra reservada del lenguaje de programación utilizado.
5. Las letras mayúsculas y minúsculas se consideran equivalentes, aunque debe preferirse letras minúsculas para variables y constantes y letras mayúsculas para las palabras reservadas del lenguaje de programación, para diferenciarlos mejor. Cabe mencionar quealgunos lenguajes son case sensitive, es decir, diferencian mayúsculas de minúsculas; por ejemplo, en los lenguajes que no son case sensitive, salario, Salario, SALARIO, SalariO, es la misma variable, mientras que en los lenguajes que sí son case sensitive, son cuatro variables diferentes.
6. El último carácter del nombre puede ser opcionalmente uno de los caracteres de declaración de tipo de valor, aunque es preferible especificar el tipo.

Para la computadora, el nombre de una constante o de una variable no es más que el identificador de una dirección o zona de memoria, en el cual se ha de almacenar y recuperar datos en el área de memoria direccionada con dicho nombre.

Los caracteres de declaración de tipo de datos se deben especificar para identificar más fácilmente qué tipo de valor almacena una constante o una variable, desafortunadamente, no en todos los lenguajes de programación se pueden utilizar los caracteres de declaración de tipo de datos, ya que el uso de ciertos caracteres puede ocasionar errores o son utilizados para otras funciones, de aquí, surge la necesidad de establecer una tabla de caracteres de declaración de tipo de datos que sea consistente y común a todos los lenguajes.

Nombramiento o declaración de variables

Las variables se pueden declarar, obviamente, dependiendo del lenguaje de programación, de dos formas:

De forma implícita: Cuando se nombran variables que no han sido explícitamente definidas o declaradas en el programa, pero que pueden utilizarse. Se dice que el lenguaje de programación es débilmente tipeado.

De forma explícita: cuando se nombran variables y son declaradas explícitamente en el programa antes de usarlas. Se dice que el lenguaje de programación es fuertemente tipeado.

Aunque el lenguaje de programación permita el uso de variables implícitas, es conveniente, como regla general, declarar o al menos definir siempre todas las variables, o cuando menos enlistarlas, de un programa, y evitar siempre que sea posible las variables implícitas.

Es una buena práctica de programación utilizar nombres de variables y constantes significativos que sugieran lo que representan, ya que eso hará más fácil y legible el programa.

También es buena práctica de programación incluir breves comentarios que indiquen cómo o para qué se utiliza la constante o la variable.

Expresión y sentencia

Una expresión es un conjunto de operadores y operandos que producen un valor o un resultado.

Por ejemplo:

“Esta operación es una ” + “expresión”
“ a esta operación se le llama “ + “concatenación”

2 + 3

Una sentencia es una línea de texto que indica una o más operaciones o acciones a realizar. La sentencia más común es la sentencia de asignación, lo cual significa que el valor que resulte de evaluar una expresión, tiene que ser asignado o almacenado en una variable especificada.

Por ejemplo:

Mensaje = “Esto es una sentencia de asignación”

Suma = 2 + 3

Nota: Si la expresión es numérica, la variable tiene que ser también numérica.
Si la expresión es alfanumérica, la variable tiene que ser también alfanumérica.
Si la expresión es lógica, la variable tiene que ser también lógica.
Si la expresión es de fecha/hora, la variable tiene que ser también de fecha/hora.

Tipos de operaciones y operadores

Operaciones alfanuméricas

Las operaciones que pueden realizar los lenguajes de programación con texto, son básicamente de concatenación, extracción y formateo.

La operación de concatenar dos o más expresiones alfanuméricas consiste en unir dichas expresiones, ya sea por medio del operador + o &, dependiendo del lenguaje de programación.

La operación de extracción consiste en obtener ciertos caracteres o un número específico de caracteres de una cadena de caracteres.

La operación de formateo consiste en obtener o convertir un valor numérico a una cadena de caracteres o viceversa.

Operaciones numéricas

Las operaciones numéricas que se realizan con números son los mismos que se realizan en las Matemáticas, además, los lenguajes de programación proporcionan funciones para realizar ciertas operaciones, para los cuales no se cuenta con algún carácter o símbolo (operador) que lo realice, tales como la raíz y la potencia.

Los tipos de operadores son:

Asociativo: Son conjuntos de operaciones o de expresiones encerrados entre paréntesis (), agrupando, de esta forma., operaciones y que indican el orden en que deben realizarse las operaciones.

Aritmético: Son operadores que realizan la operación matemática indicada, así, + para sumar, – para restar, * para multiplicar, / para dividir.

Reglas de prioridad

Para realizar un conjunto de operaciones en una misma expresión, es decir, que dicha expresión tenga dos o más operandos u operadores, se requieren de reglas que permitan determinar con lógica el orden de las operaciones.

A estas reglas se les denominan reglas de prioridad, y son:

1. En las operaciones que estén encerradas entre paréntesis anidados (es decir, interiores unos a otros), las operaciones o expresiones más internas se realizan o evalúan primero.
2. Operadores de potencia y raíz.
3. Operadores de producto (multiplicación) y cociente (división).
4. Operadores de división entera y resto de división (módulo).
5. Operadores de adición (suma) y sustracción (resta).
6. En caso de coincidir varios operadores de igual prioridad en una expresión, encerrada o no entre paréntesis, el orden de prioridad en este caso, es de izquierda a derecha.

Operaciones de relación y lógicas

Las operaciones de relación y lógicas tienen como característica que el valor que producen es un valor de verdad o de falsedad (verdadero o falso), esto quiere decir que una proposición es un enunciado que afirma o niega algo y puede ser calificado como falso o verdadero.

Operadores relacionales

Son operadores que permiten comparar dos valores, sean numéricos, alfanuméricos, booleanos o de fecha/hora. El resultado es un valor de verdad, que puede ser verdadero o falso.

Los operadores relacionales son los siguientes:

Igual	=
Diferente	<>
Mayor que	>
Menor que	<
Mayor o igual que	>=
Menor o igual que	<=

Operadores lógicos

Son operadores que permiten realizar operaciones lógicas de conjunción, disyunción, negación, implicación y equivalencia. El resultado obtenido es un valor de verdad que puede ser verdadero o falso.

Los operadores son, en orden de jerarquía (prioridad):

Negación	
A	B
V	F
F	V

NOT (Negación): la negación es la proposición simple que resulta de aplicar este operador lógico NO a otra proposición. Su símbolo es . Este operador convierte una proposición en su opuesto

AND (Conjunción): la conjunción es una proposición compuesta que resulta de aplicar el operador lógico Y a un par de proposiciones, su símbolo es . En esta combinación, C puede ser verdadero, siempre que A y B sean verdaderos.

OR (Disyunción): la disyunción es una proposición compuesta que resulta de aplicar el operador lógico O a un par de proposiciones, su símbolo es . En esta combinación, C puede ser verdadero en los tres casos en que cualquiera de las proposiciones A o B sean verdaderos.

IMP (Implicación o condicional): la implicación es una proposición compuesta que resulta de aplicar el operador lógico SI … ENTONCES… a un par de proposiciones. Su símbolo es . la implicación sólo es falsa cuando la hipótesis (A) es verdadera y la tesis (B) es falsa. En una implicación, la primera preposición (A) se llama hipótesis o antecedente y la segunda preposición (B) se llama tesis, consecuente o conclusión.

EQV (Equivalencia, doble implicación o bicondicional): la equivalencia es una proposición compuesta que resulta de aplicar el operador lógico … si y sólo si …. Su símbolo es . la equivalencia sólo es verdadera cuando las dos proposiciones que la forman tienen el mismo valor de verdad, es decir, las dos proposiciones son verdaderas o las dos proposiciones son falsas.

XOR: es lo opuesto a EQV, es decir es la negación de EQV.

Nota: los tres últimos operadores lógicos, no en todos los lenguajes de programación se encuentran.

Las dos columnas de las tablas corresponden a todas las posibles combinaciones de verdad y falsedad que se pueden realizar con dichos valores.

Tipos de Problemas

Características de los problemas

En programación estructurada, existen tres estructuras de control básicas para describir un proceso en particular. Durante el bosquejo de la solución y el algoritmo, se identifica cuál de las tres estructuras es el adecuado para resolver un problema de acuerdo a sus características.

Dichas estructuras de control permiten el diseño de un programa de una manera más entendible y facilitan la codificación.

Estas estructuras de control muestran acciones incondicionales, acciones repetitivas y acciones que ocurren sólo cuando se presentan ciertas condiciones. Dichas estructuras de control se pueden utilizar ya sea en forma individual o en combinación con alguna otra, dependiendo del problema a solucionar.

Tipos de estructuras

Existen tres tipos de estructuras de control para describir un proceso: estructuras de secuencia, estructuras de decisión o selección y estructuras de repetición o iteración.

La estructura de secuencia

Una estructura de secuencia, o estructura secuencial, es un solo paso o acción a realizarse, incluida en un proceso. Cada paso no depende de la existencia de alguna condición para describir un proceso.

Ejemplo:

```
Algoritmo EstructuraSecuencial

    Entrar a librería
    Escoger el libro deseado
    Llevar el libro deseado al mostrador de salida
    Pagar el libro
    Obtener el recibo
    Salir de librería

FinAlgoritmo
```

Este ejemplo muestra una secuencia incondicional de seis pasos o acciones. Ningún paso contiene alguna decisión o condición que determine la realización del siguiente paso. Los pasos se efectúan en un orden lógico, por ejemplo, tiene poco sentido pagar por un libro no deseado.

Por consiguiente, en el procedimiento se muestra el orden correcto de las acciones o pasos a realizar, siempre se lleva a cabo esta secuencia de seis pasos, uno después del otro, sin ninguna decisión sobre el orden o relacionada con condiciones

La estructura de selección o de decisión

Este tipo de estructura se utiliza cuando se tiene que decidir o seleccionar qué pasos se han de realizar dependiendo de condiciones especificadas. Para esto, primero se evalúa la condición y después se toma la decisión de realizar la o las acciones asociadas con esta condición. Una vez determinada la condición, las acciones que se realicen son incondicionales.

Esta estructura de control tiene tres alternativas, que son:

Alternativa simple

La decisión simple se usa para determinar una condición, si dicha condición se cumple, se ejecuta la o las acciones que están dentro de la estructura de decisión simple, si no, el control del programa pasa a la siguiente instrucción inmediata fuera de la estructura.

Ejemplo.

```
Algoritmo AlternativaSimple

    Entrar a librería
    Buscar el libro deseado

    Si se encuentra el libro deseado Entonces

        Llevar el libro deseado al mostrador de salida
        Pagar el libro
        Obtener el recibo

    FinSi

    Salir de librería

FinAlgoritmo
```

Alternativa doble

En la decisión doble se evalúa una condición, si dicha condición se cumple, se ejecuta la o las acciones que estén dentro de la parte SI, si no, se ejecuta la o las acciones que estén dentro de la parte SiNo, en cualquiera de los dos casos, al cumplirse o no la condición, el control del programa se pasa a la siguiente instrucción inmediata fuera de la estructura.

```
Algoritmo AlternativaDoble

    Entrar a librería
    Buscar el libro deseado

    Si se encuentra el libro deseado Entonces

        Llevar el libro deseado al mostrador de salida
        Pagar el libro
        Obtener el recibo

    SiNo

        No llevar libros al mostrador de salida

    FinSi

    Salir de librería

FinAlgoritmo
```

En este ejemplo se ilustra que, al ir a la librería, es posible que tenga o no tenga en existencia el libro deseado (el libro que se desea comprar), en este caso, se tienen dos condiciones: encontrar el libro y no encontrar el libro.

La parte Si contiene tres declaraciones separadas en secuencia, acciones que han de realizarse si se encuentra el libro deseado; mientras que en la alternativo doble se da otra opción, estableciendo de otro modo o de lo contrario, en la parte SiNo, que contiene sólo una acción, que ha de realizarse si no se encuentra el libro deseado.

Alternativa múltiple

Con frecuencia, es necesario que existan más de dos elecciones u opciones posibles. Este problema se podría resolver por estructuras de alternativa simple o doble anidadas o en cascada. Sin embargo, si el número de opciones o alternativas es grande, puede plantear serios problemas de estructura de algoritmo y naturalmente de legibilidad en el programa.

La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores u opciones distintas (1, 2, 3, 4, … n), según se elija uno de estos valores u opciones en la condición, se realizará una de las n acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los n posibles, dependiendo de la condición.

Por ejemplo:

```
Algoritmo LetrasVocales

Segun opción Hacer

        Opción “A”:
                Escribir “Es la letra A”
        Opción “E”:
                Escribir “Es la letra E”
        Opción “I”:
                Escribir “Es la letra I”
        Opción “O”:
                Escribir “Es la letra O”
        Opción “U”:
                Escribir “Es la letra U”
        De Otro Modo:
                Escribir “No es una vocal”

Fin Segun

FinAlgoritmo
```

La estructura de repetición o de iteración

Esta estructura de control se utiliza para repetir actividades o acciones de rutina, esta repetición se realiza mientras existan ciertas condiciones o hasta que estas condiciones se presentan. A la estructura de control de repetición o iteración se le conoce también como bucle, lazo, ciclo o rizo.

Esta estructura de control tiene tres ciclos, los cuales son:

Ciclo Para o Desde

En muchas ocasiones se conoce de antemano el número de veces que se desea ejecutar una serie de acciones o pasos de una repetición o iteración. En este caso, se utiliza el ciclo Para o Desde.

La estructura Para ejecuta las acciones contenidas en un bucle un número especificado de veces y de modo automático controla el número de iteraciones o repeticiones a través del bucle.

Por ejemplo:

```
Algoritmo BuscarLibro
```

```
    Entrar a la librería

    Para iniciar a examinar libros Hasta cinco libros Con Paso un libro a la vez Hacer

        Leer el título del libro
        Hojearlo
        Regresarlo al estante

    Fin Para

    Salir de la librería

FinAlgoritmo
```

En el ejemplo anterior se ilustra que se repiten cinco veces los pasos contenidos dentro del bucle, e inmediatamente después de ejecutarse dichas iteraciones, se sigue con el siguiente paso o instrucción fuera del ciclo.

Ciclo Mientras

Existen muchas situaciones en las que se desea que un bucle se ejecute una y otra vez mientras que la condición sea verdadera.

El ciclo Mientras es aquel ciclo en que el bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta, primero se evalúa la condición, si se evalúa falsa el bucle no ejecuta acción alguna dentro de la repetición y sigue o continúa con la siguiente acción o instrucción fuera del ciclo.

Si la condición es verdadera, se ejecuta el ciclo una y otra vez mientras que la condición sea verdadera.

Ejemplo:

```
Algoritmo BuscarLibro2

    Entrar a la librería

    Mientras se examinan libros Hacer

        Leer el título del libro
        Si el título del libro suena interesante Entonces
            Tomar el libro y hojearlo
            Buscar el precio del libro

            Si la decisión es comprar el libro Entonces

                Llevar el libro a la pila de libros para llevar

            SiNo

                Regresar el libro al estante

            FinSi

        SiNo

            Continuar examinando libros

        Fin Si

    Fin Mientras

FinAlgoritmo
```

Ciclo Repetir

Existen muchas situaciones en las que se desea que un ciclo se ejecute al menos una vez antes de comprobar la condición de repetición. En el ciclo Mientras, si la condición es inicialmente falsa, el bucle no se ejecutará, por ello, se necesita otro tipo de ciclo.

El ciclo Repetir se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle.

El ciclo Repetir se repite mientras el valor de la condición sea falso, lo opuesto al ciclo Mientras.

Ejemplo:

```
Algoritmo BuscarLibro3

    Entrar a la librería

    Repetir

        Leer el título del libro
        Si el título del libro suena interesante Entonces
            Tomar el libro y hojearlo
            Buscar el precio del libro

            Si la decisión es comprar el libro Entonces

                Colocar el libro en la pila de libros para llevar

            SiNo

                Regresar el libro al estante

            FinSi

        SiNo

            Continuar examinando libros

        FinSi

    Hasta Que se termine de examinar libros

FinAlgoritmo
```

En los ejemplos de los dos ciclos anteriores, Mientras y Repetir, se observa que se describen cero, uno o más iteraciones o repeticiones para describir las acciones o las instrucciones a realizar. Los pasos adicionales, dentro de la iteración, dan instrucciones sobre qué hacer cuando se cumplan o no ciertas condiciones, en este caso, cuando se encuentran o no los libros deseados.

La repetición de este proceso continúa siempre y cuando exista la condición de examinar más libros.

La iteración implica repetición, es decir, significa que las instrucciones que integran el ciclo Mientras o el ciclo Repetir serán repetidas, ¿cuántas veces?, esto depende de la condición; a diferencia del ciclo Para, que ya se sabe previamente cuántas veces se repetirán las acciones o instrucciones.

Las diferencias de los ciclos Mientras y Repetir se concretan en:

- El ciclo Mientras termina cuando la condición es falsa, mientras que el ciclo Repetir termina cuando la condición es verdadera.
- En el ciclo Repetir, el bucle se ejecuta siempre al menos una vez; por el contrario, el ciclo Mientras permite la posibilidad de que el bucle pueda no ser ejecutado.
- Para usar el ciclo Repetir, se debe tener la seguridad de que el bucle, bajo cualquier condición, se ejecutará al menos una vez.

Programación

La programación es el proceso que consiste de una serie de pasos involucrados en la solución de un problema. La actividad de la programación es conceptual y su campo de acción se centra en definir y comprender el problema clarificando cualquier ambigüedad o duda en el enunciado del mismo. Posteriormente, se decide cómo solucionarlo, es decir, se bosqueja un esquema de la solución en alguna notación adecuada.

Con frecuencia, el término programación es empleado como sinónimo de codificación, lo cual es un error por lo siguiente: codificar es una etapa posterior a la de programar, y consiste en el proceso de escribir enunciados o sentencias en un lenguaje de programación adecuado. Por lo tanto, primero se programa la solución del problema y después se procede a traducir la solución a algún lenguaje de programación.

Programar, como ya se mencionó, es el proceso constituido por varias etapas, en donde se define con claridad el problema que se va a resolver por medio de una computadora, así como el procedimiento mediante el cual la computadora llegará a la solución deseada.

Las fases que integran la programación varían de acuerdo con los estilos y los enfoques de un programador a otro, incluso, de un autor a otro; no obstante, las siguientes son válidas:

1. Definición del problema.
2. Planeación o bosquejo de la solución.
3. Algoritmo.
4. Diagrama de flujo.
5. Codificación.
6. Verificación y depuración.
7. Prueba y validación.
8. Documentación.
9. Mantenimiento.

Que ya fueron analizadas en la Metodología para la Solución de Problemas.

Pseudocódigo

Como se ha mencionado anteriormente, las primeras fases del proceso de programación son la definición del problema y posteriormente se obtiene la representación adecuada de la descripción de la solución. Esta descripción puede ser definida por representaciones adecuadas de símbolos a través de ciertos lenguajes de programación, que funcione como un vehículo descriptor y como un modelo de representación dada a la solución. Este lenguaje de programación deberá denotar independientemente con respecto a alguna computadora o lenguaje de programación en específico y debe ser capaz de representar o expresar cualquier idea computacional.

Este lenguaje recibe el nombre de Pseudocódigo o Español Estructurado.

Tanto el siguiente ejemplo, como los ejemplos presentados de las estructuras de control en el capítulo anterior, son ejemplos de Pseudocódigo.

```
Algoritmo SumarNNumerosPares

    Definir Numero1, Suma, I Como Entero;

    Numero1 <- 0;
    Suma <- 0;

    Escribir "Introduce un número entero hasta donde se desea la suma de números enteros pares: ";
    Leer Numero1;

    Para I = 2 Hasta Numero1 Con Paso 2 Hacer

        Suma <- Suma + I;

    Fin Para
```

```
    Escribir "La suma de los números enteros pares es: ", Suma;

FinAlgoritmo
```

El Pseudocódigo es un conjunto de oraciones sencillas que se utilizan para representar y escribir un algoritmo, tal como se aprecia en el ejemplo anterior.

El Pseudocódigo es una forma de representar y escribir algoritmos. El Pseudocódigo es una herramienta muy poderosa para los programadores profesionales y, en particular, para los que no tengan experiencia en el diseño de programas.

Algunas de sus características son:

- ✓ Es informal e incluye pocas o ninguna regla gramatical o de sintaxis estricta.
- ✓ Es sumamente manejable.
- ✓ Se encuentra en un punto intermedio entre la representación de un lenguaje natural (ya sea el castellano o el inglés) y la representación del lenguaje de programación (BASIC, Pascal, C/C++, C#, Java, Fortran, JavaScript, PHP, Perl, Python, TypeScript, SQL, etc.).
- ✓ Su elaboración y características estructurales se acercan a cualquier lenguaje de programación deseado, así se facilita el siguiente paso de la traducción, es decir, la codificación a dicho lenguaje de programación.
- ✓ Es flexible y no se limita a la sintaxis restrictiva de algún lenguaje de programación.

Con un adecuado uso de sangrías, o identación, y de espaciados, se indican claramente las relaciones entre los distintos enunciados y permitirá obtener una mayor visión de cómo está desarrollado la estructura y la lógica del programa.

Diseño de programas

Un programa debe diseñarse de tal modo que sea legible y comprensible, para sí poder realizar las modificaciones posteriores que sean necesarias o darle mantenimiento de manera más fácil y rápida.

Estructura de un programa

Todo programa debe estar separado en secciones que sean fácilmente reconocibles, a continuación, se muestra una estructura de programa como ejemplo:

- Nombre del archivo, programa o módulo.
- Una sección para los pasos que sean necesarios incluir, listados en el Capítulo II, en la fase de la codificación.
- Una sección para definir las funciones de biblioteca que disponga los lenguajes de programación y que sean necesarios utilizar.
- Una sección para definir tipos de datos propios.
- Una sección para declarar explícitamente variables.
- Una sección para declarar explícitamente constantes.
- Una sección para definir funciones.
- Una sección para definir procedimientos o subrutinas.
- Especificar el inicio del programa principal.
- La codificación (programa principal).
- Especificar el fin del programa principal.

Obviamente la estructura anterior variará de acuerdo con el lenguaje de programación que se utilice. La presentación final de un programa viene influida no sólo por los comentarios o por la estructura presentada, sino por otra consideración de tipo práctico muy importante y que se debe tomar en cuenta.

Aunque existen lenguajes de programación, como BASIC, Fortran, C/++, que tienen reglas estrictas de escritura, se debe tender a escribir los programas, en cualquier lenguaje de programación, con el estilo de escritura utilizando sangrado o identación, como en el Pseudocódigo, es decir, dar sangrías a instrucciones anteriores a las estructuras. Las sangrías son de gran ayuda al revelar la estructura lógica del programa.

Lenguajes de programación

Un lenguaje de programación es el software de base que proporciona un conjunto de vocablos y reglas, que se necesitan para crear el software de aplicación.

El lenguaje de programación más elemental es el lenguaje de máquina (lenguaje de bajo nivel), que es una colección o conjunto de instrucciones en código binario muy detalladas y difíciles de descifrar.

Existen, además del lenguaje máquina, otros, que se aproximan más a los lenguajes humanos, por lo regular, en inglés, por lo cual, son más sencillos de manejar, a estos lenguajes de programación se les llama lenguajes de alto nivel.

Debido a que la computadora no entiende lo que significa algún comando o instrucción en un lenguaje de programación de alto nivel, necesita hacer una traducción a instrucciones binarias, esta traducción se realiza de dos formas: por medio de un compilador, que convierte las instrucciones del lenguaje de programación de alto nivel a código binario de una vez; o por medio de un intérprete que traduce las instrucciones del lenguaje de programación de alto nivel a código binario cada vez que se ejecuta el programa.

Intérpretes

Los programas intérpretes traducen un programa escrito en un lenguaje de programación de alto nivel, el cual es llamado programa fuente, a un lenguaje máquina, al cual s ele llama programa objeto, de la siguiente manera: en el programa intérprete se lee una instrucción del programa fuente; analiza su sintaxis y su semántica; si éstas están correctas las traduce a una o varias instrucciones del lenguaje máquina para ser ejecutadas; procede después a leer otra instrucción. De esta manera, las instrucciones del programa objeto, en lenguaje máquina, no se almacenan, sino que se van ejecutando conforme se va generando, es decir, conforme se va leyendo el programa fuente.

Algunos intérpretes permiten visualizar inmediatamente los resultados de un programa escrito en algún lenguaje de programación de alto nivel; indican si están correctamente escritas las instrucciones; sin embargo, su desventaja es la de disponer del intérprete para poder ejecutar los programas.

Compiladores

Los programas compiladores son programas que se encargan de traducir programas escritos en un lenguaje de programación de alto nivel, el cual es llamado programa fuente, a un lenguaje máquina, al cual s ele llama programa objeto; sin embargo, existen una serie de diferencias con respecto a los programas intérpretes: los compiladores realizan la traducción completa del programa fuente; analizan la sintaxis de las instrucciones de todo el programa; en caso d no existir errores, generan otro archivo con las instrucciones en lenguaje máquina, que es el programa objeto.

Algunos compiladores dejan en los programas objeto el código necesario para su ejecución en la computadora; otros necesitan un proceso posterior para dejar el código en ejecutable, a este proceso se le llama linkeditación (enlace), en el cual, son agregados códigos de utilería para la interpretación de algunas instrucciones y que puedan éstas ser ejecutadas.

El programa original escrito en un lenguaje de programación de bajo nivel, de medio nivel o de alto nivel, se le denomina programa fuente, y el programa traducido a lenguaje máquina se le conoce como programa objeto, que es una forma intermedia que finalmente es convertido en un programa ejecutable, ya directamente ejecutable por la computadora.

En resumen, el proceso es el siguiente:

Programa fuente	→	.bas, .pas. .c. .for, (instrucciones escritas en algún lenguaje de programación de alto nivel)
Programa objeto	→	.obj (ya directamente ejecutable)
Programa ejecutable	→	.exe, .exec (programas ejecutables sin necesidad de traducciones)

Lenguajes de programación de alto nivel

Después de definir claramente el problema de organizar una solución y plantear los detalles del algoritmo paso a paso, puede entonces llevarse a cabo la codificación.

Aunque el algoritmo ha sido probado y considerado en principio como correcto, no es ejecutable por una computadora, por lo que es necesario refinarlo aún más. El proceso de refinamiento consiste en acercar lo más posible el algoritmo a un programa fuente escrito en algún lenguaje de programación de alto nivel en particular, por ejemplo: BASIC, Pascal, C/C++, C#, Java, Fortran, Python, PHP, JavaScript, Perl, TypeScript, SQL, etc.

La elección del lenguaje de programación de alto nivel a utilizar, dependerá de las siguientes consideraciones:

1. La naturaleza del problema.
2. Los lenguajes de programación de alto nivel con que se dispone.

3. Las características y limitaciones de los recursos con que cuenta la computadora, es decir, los soportes con que cuenta de entrada, de salida y la capacidad de la memoria y del procesador.

Existen ciertos lenguajes de programación de alto nivel de uso o de propósito general, otros son específicos para ciertos tipos de problemas, existen algunos más que sólo se ejecutan en ciertos tipos de computadora.

Los lenguajes de programación de alto nivel más comunes son: Fortran, Algol, Ada, Logo, Progress, Pilot, Forth, Modula2, Cobol, Lisp, JavaScript, PHP, Perl, Python, Snobol, BASIC, PL/1, APL, APT, C, C++, Pascal, DBase, Clipper, TypeScript, SQL, entre otros; y los llamados de cuarta generación: Visual Basic, Visual C, Delphi, etc.

Como se observa, existen centenares de lenguajes de programación y una gran variedad o versiones de los lenguajes más utilizados, pero los procesos fundamentales y las estructuras básicas son similares y constituye un fundamento conceptual el hecho de aprender cualquier lenguaje de programación.

Al aprender a programar se desarrollan dos conocimientos muy importantes:

- Se aprende la sintaxis, las palabras claves o reservadas, gramática y puntuación del lenguaje de programación en sí. Se conoce lo que cada comando, instrucción y función realiza y, por lo tanto, cómo utilizarlos.
- También se descubre la lógica de la programación, cómo realizar una tarea o un proceso utilizando el lenguaje de programación, éste es un concepto que se puede aplicar a cualquier lenguaje de programación.

Una vez que se conoce la lógica para programar en un lenguaje de programación, se puede aprender cualquier otro lenguaje de programación, aprendiendo tan sólo sus palabras reservadas y su sintaxis.

Básicamente se requieren ambos conocimientos para programar.

Los lenguajes de programación más utilizados

Los lenguajes de programación de alto nivel más usados o más populares de acuerdo con varias fuentes, definiendo diversos criterios como: usabilidad, facilidad, curva de aprendizaje, demanda laboral, etc., a finales de 2023, son:

1	Python	JavaScript	Python	JavaScript	Java	JavaScript	JavaScript
2	JavaScript	Python	C	Python	C	Python	HTML
3	Java	SQL	C++	HTML	Python	Swift	CSS
4	HTML	TypeScript	Java	PHP	C++	TypeScript	SQL
5	CSS	Java	C#	CSS	Visual Basic .NET	Kotlin	Python
6	SQL	C#	JavaScript		C#	Rust	TypeScript
7	C#	C++	Visual Basic .NET		JavaScript		Java
8	C	PHP	PHP		PHP		Bash/Shell
9	C++	Go	Assembler		SQL		C#
10	TypeScript	Rust	SQL		Objective-C		C++
11	PHP						PHP
12	R						C
13	Bash/Shell						PowerShell
14	Swift						Go
15	Objective-C						Rust

Como se observa en la tabla anterior, los dos lenguajes más utilizado o más populares son: Python y JavaScript; después, PHP; posteriormente, Java, SQL, C#, C++; después, TypeScript, C; seguidamente, HTML, CSS, Rust.

Lo anterior no significa que un lenguaje de programación sea mejor que otro, cada lenguaje de programación ha sido desarrollado o adaptado a situaciones y objetivos diferentes.

Qué se requiere para programar

Para programar aplicaciones en algún lenguaje de programación, se requiere, básicamente, de un editor de textos y un compilador o intérprete, según sea el caso.

Editores de texto

Los editores de texto son herramientas que se utilizan para escribir y editar código en algún lenguaje de programación para llevar a cabo el desarrollo de software o de aplicaciones.

Algunos editores de texto cuentan con diferentes funciones como el autocompletado, el resaltado de sintaxis y la programación es mucho más fácil y rápida en comparación con los editores de texto que se utilizaban anteriormente.

IDE (Integrated Development Enviroment = Entorno de Desarrollo Integrado)

Es una aplicación que proporciona soporte integral para facilitar al desarrollador o programador la escritura o desarrollo de software.

Un IDE consiste en un editor de textos, para la escritura de código fuente; herramientas de construcción, compilador o intérprete; y depurador. Asimismo, la mayoría de los IDE tienen una característica conocida como IntelliSense, que permite autocompletar de forma inteligente el código, al predecir los comandos o las instrucciones a utilizar.

Algunos IDE están dedicados o diseñados específicamente para un lenguaje de programación; otros, son multilenguaje, permitiendo el uso de distintos lenguajes de programación con el mismo IDE.

Paradigmas de programación

Un paradigma, o modelo, de programación define la forma en que un programador ve la ejecución de un programa. Los paradigmas de programación están comúnmente asociados a una determinada escuela o pensamiento de ingeniería de software y hasta son representados por un conjunto de lenguajes de programación, por ejemplo, BASIC, Pascal o C son lenguajes imperativos; Smalltalk y Java se asocian con el paradigma de programación orientada a objetos; mientras que Haskell y Scheme se asocian con el de programación funcional. Otros lenguajes de programación, tales como Common Lisp y Python, están diseñados para adaptarse al uso de diferentes paradigmas, es decir, son multiparadigmas.

Programación imperativa

La programación imperativa es una forma de programar que describe el procesamiento en términos de un estado del programa, el contenido de la memoria, por ejemplo, y un conjunto de instrucciones, órdenes o comandos, por eso se llama imperativa, que cambian ese estado.

Los primeros lenguajes de programación se basaron em el principio de la programación imperativa, siguiendo el mecanismo de ejecución similar al de una receta, donde cada paso es una instrucción y los demás elementos representan el estado del programa. Estos lenguajes fueron evolucionando con la incorporación de estructuras de control de flujo, variables, definición de tipos, etc., hasta derivar en los lenguajes estructurados que se utilizan hoy en día.

Programación procedural

Se basa en la utilización de procedimientos o subrutinas, reunidas en módulos, para particionar o dividir un programa y para limitar el alcance en la visibilidad de los datos y de las variables.

Los programas procedurales tienen múltiples niveles o alcances, con procedimientos definidos dentro de otros procedimientos. Cada alcance puede contener variables que no son visibles desde un alcance exterior.

Programación estructurada

Como se comentó anteriormente, la programación estructurada podría verse como un subconjunto o subdisciplina de la programación procedural, pensada con la intención de facilitar el mantenimiento de un programa a partir de la escritura de código simple, entendible y modularizado o particionado.

Se han desarrollado técnicas o metodologías para escribir programas estructurados. Dos de las metodologías más utilizadas son:

1) Alinear estructuras de control de flujo de los programas.
2) Dividir los programas en subsecciones o módulos, cada uno con un único punto de entrada y un único punto de salida.

La programación estructurada recomienda utilizar estructuras de control de flujo simples y organizadas en forma jerárquica para controlar el flujo de un programa; asimismo, dividir en grandes bloques de código en subrutinas, por ejemplo, funciones y procedimientos, que sean lo suficientemente pequeños para resultar fácilmente entendibles

Programación declarativa

Muestra a la computadora un conjunto de condiciones y le permite averiguar cómo satisfacerlas, la programación declarativa describe relaciones entre variables, en términos de funciones o reglas de inferencia.

Programación funcional

Trata a la programación como la evaluación de funciones. Enfatiza la evaluación de expresiones funcionales, en lugar de la ejecución de comandos

Programación lógica

Se especifica un conjunto de atributos que debe reunir la solución de un problema, el proceso de resolución de problemas en este paradigma se basa en hechos más reglas para obtener resultados, lo cual se hace a través de principios formales de la lógica.

Programación orientada a objetos

Hace que los datos dejen de ser elementos pasivos, definidos por relaciones o utilizados por funciones y procedimientos, para convertirse en elementos activos que interactúan entre sí.

Es una abstracción y generalización de la programación imperativa. Involucra un estado y un conjunto de operaciones para transformar ese estado. Involucra colecciones de objetos, cada uno con un estado y un conjunto de operaciones transformarlo, el énfasis pasa de describir el flujo del control a describir la interacción entre objetos.

El fundamento básico de la programación orientada a objetos es el agrupamiento de los datos y las acciones en un único objeto

Programación paralela

La programación paralela o programación concurrente permite la ejecución de operaciones en forma concurrente, ya sea en forma real o aparente en un solo programa a lo largo de un conjunto de instrucciones de sistemas. En este caso, se utiliza el término computación distribuida.

Ejemplos de Programación

En esta parte, se dan algunos ejemplos de prácticas de programación; un aspecto de gran importancia en la programación, es el análisis del problema y el manejo de la información, es decir, determinar qué datos necesita el programa y cómo se llevará a cabo el proceso.

Solo a través de una comprensión clara de lo que se desea que realice el programa, se podrá desarrollar una solución razonable y adecuada para cualquier problema.

Problemas con estructura de secuencia

Ejemplo 1

Realizar un programa que realice las operaciones básicas aritméticas con dos números enteros.

Paso 1

Realizar un programa que realice las operaciones básicas aritméticas con dos números enteros.

Paso 2

Entrada	→	el valor de un número entero y el valor de otro número entero
Salida	→	el resultado de las operaciones básicas aritméticas con dichos números

Datos adicionales	→	sin datos adicionales
Solución	→	el resultado de las operaciones básicas aritméticas a realizar con dichos números

Paso 3

Pedir datos	→	el valor de un número y el valor de otro número
Desplegar datos	→	los valores proporcionados
Evaluar condiciones	→	no se requiere evaluar condiciones
Ejecutar operaciones	→	Suma <- Numero1 + Numero2 Resta <- Numero1 – Numero2 Producto <- Numero1 * Numero2 División <- Numero1 / Numero2
Desplegar resultados	→	los resultados de las operaciones básicas aritméticas realizadas con dichos números

```
Algoritmo OperacionesBasicascon2Numeros

    Definir Numero1, Numero2 Como Entero;
    Definir Suma, Resta, Producto, Division Como Entero;

    Escribir "Introduce el primer número entero: ";
    Leer Numero1;
    Escribir "Introduce el segundo número entero: ";
    Leer Numero2;

    Escribir "Los números enteros introducidos son: ", Numero1, " y ", Numero2;

    Suma <- Numero1 + Numero2;
    Resta <- Numero1 - Numero2;
    Producto <- Numero1 * Numero2;
    Division <- Numero1 / Numero2;

    Escribir "La suma de ", Numero1, " y ", Numero2, " es: ", Suma;
    Escribir "La resta de ", Numero1, " y ", Numero2, " es: ", Resta;
    Escribir "El producto de ", Numero1, " y ", Numero2, " es: ", Producto;
    Escribir "La división de ", Numero1, " y ", Numero2, " es: ", Division;

FinAlgoritmo
```

Ejercicio 1

Crear un programa que muestre el mensaje Hola, Mundo

```
Algoritmo HolaMundo

    //Introducción a la Programación
    //Programa 1: EstSec1.psc
    //Hola, Mundo.
    //Programa que muestra el mensaje "Hola, Mundo."
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Mostrar, desplegar o imprimir mensaje
    Escribir "Hola, Mundo.";

FinAlgoritmo
```

Ejercicio 2

Escribir un programa que muestre el mensaje Hola, Mundo. Bienvenido/a a la Introducción a la Programación, de diferentes formas.

```
Algoritmo Bienvenido

    //Introducción a la Programación
    //Programa 2: EstSec2.psc
    //Hola, Mundo. Bienvenido a la Introducción a la Programación
    //Programa que muestra el mensaje "Hola, Mundo. Bienvenido/a a la Introducción a la Programación", en diferentes formas
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Primera forma de mostrar o imprimir el mensaje
    Escribir "Hola, Mundo. Bienvenido/a a la Introducción a la Programación.";
    Escribir "";

    //Segunda forma de mostrar o imprimir el mensaje
    Escribir "Hola, Mundo.";
    Escribir "Bienvenido/a a la Introducción a la Programación.";
    Escribir "";

    //Tercera forma de mostrar o imprimir el mensaje
    Escribir "Hola, Mundo.", "Bienvenido/a a la Introducción a la Programación.";

FinAlgoritmo
```

Ejercicio 3

Realizar un programa que realice la suma, resta, multiplicación y división de dos números enteros y muestre el resultado.

```
Algoritmo SumaRestaMultiplicacionDivisionConDosNumerosEnteros

    //Introducción a la Programación
    //Programa 3: EstSec3.psc
    //Suma, resta, multiplicación y división con dos números enteros dados
    //Programa que muestra el resultado de la suma, resta, multiplicación y división de dos números enteros dados
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Numero1, Numero2, Suma, Resta, Multiplicacion, Division Como Entero;

    //Asignar valores a las variables de Numero1 y Numero2
    Numero1 <- 8;
    Numero2 <- 16;

    //Realizar las operaciones
    Suma <- Numero2 + Numero1;
    Resta <- Numero2 - Numero1;
    Multiplicacion <- Numero2 * Numero1;
    Division <- Numero2 / Numero1;

    //Mostrar los resultados
    Escribir "La suma de ", Numero2, " + ", Numero1, " es: ", Suma;
    Escribir "La resta de ", Numero2, " - ", Numero1, " es: ", Resta;
    Escribir "La multiplicación de ", Numero2, " * ", Numero1, " es: ", Multiplicacion;
    Escribir "La división de ", Numero2, " / ", Numero1, " es: ", Division;

FinAlgoritmo
```

Ejercicio 4

Hacer un programa que realice la suma, resta, multiplicación y división de dos números reales que el usuario introduzca.

```
Algoritmo SumaRestaMultiplicacionDivisionConDosNumerosRealesLeidos

    //Introducción a la Programación
    //Programa 4: EstSec4.psc
    //Suma, resta, multiplicación y división con dos números reales leídos
    //Programa que muestra el resultado de la suma, resta, multiplicación y división de dos números reales leídos
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Numero1, Numero2, Suma, Resta, Multiplicacion, Division Como Real;

    //Leer Numero1 y Numero2
    Escribir "Introduce el primer número: ";
    Leer Numero1;
    Escribir "Introduce el segundo número: ";
    Leer Numero2;

    //Realizar las operaciones
    Suma <- Numero2 + Numero1;
    Resta <- Numero2 - Numero1;
    Multiplicacion <- Numero2 * Numero1;
    Division <- Numero2 / Numero1;

    //Mostrar los resultados
    Escribir "La suma de ", Numero2, " + ", Numero1, " es: ", Suma;
    Escribir "La resta de ", Numero2, " - ", Numero1, " es: ", Resta;
    Escribir "La multiplicación de ", Numero2, " * ", Numero1, " es: ", Multiplicacion;
    Escribir "La división de ", Numero2, " / ", Numero1, " es: ", Division;

FinAlgoritmo
```

Ejercicio 5

Realizar un programa que calcule el diámetro, la circunferencia y el área de un círculo, solicitando al usuario que introduzca el radio.

```
Algoritmo DiametroCircunferenciaAreaCirculo

    //Introducción a la Programación
    //Programa 5: EstSec5.psc
    //Cálculo de Diámetro, Circunferencia y Área
    //Programa que muestra el Diámetro, la Circunferencia y el Área de un Círculo, solicitando al usuario que introduzca el Radio
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Radio, Diametro, Circunferencia, Area Como Real;

    //Leer el Radio
    Escribir "Introduce el radio del círculo";
    Leer Radio;

    //Realizar cálculos
```

```
    Diametro <- 2 * Radio;
    Circunferencia <- 2 * Pi * Radio;
    Area <- PI * (Radio * Radio);

    //Mostrar resultados
    Escribir "El Diámetro del círculo es: ", Diametro;
    Escribir "La Circunferencia del círculo es: ", Circunferencia;
    Escribir "El Área del círculo es: ", Area;

FinAlgoritmo
```

Ejercicio 6

Realizar un programa que convierta grados Celsius o Centígrados a grados Fahrenheit y a Kelvin.

```
Algoritmo ConvertirCelsisusaFahrenheitKelvin

    //Introducción a la Programación
    //Programa 6: EstSec6.psc
    //Convertir grados Celsius o Centígrados a Kelvin y a grados Fahrenheit
    //Programa que convierte una temperatura en grados Celsius o Centígrados a Kelvin y a grados Fahrenheit
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Celsius, Kelvin, Fahrenheit Como Real;

    //Leer la temperatura en grados Celsius
    Escribir "Introduce la temperatura en grados Celsius: ";
    Leer Celsius;

    //Realizar operaciones
    Fahrenheit <- (9 / 5) * Celsius + 32;
    Kelvin <- Celsius + 273.15;

    //Mostrar los resultados
    Escribir Celsius, " grados Celsius, es equivalente a ", Fahrenheit, " grados Fahrenheit";
    Escribir Celsius, " grados Celsius, es equivalente a ", Kelvin, " Kelvin";

FinAlgoritmo
```

Ejercicio 7

Hacer un programa que solucione un sistema de ecuaciones lineales de dos variables.

```
Algoritmo SolucionSistemaEcuacionesLinhealesDosVariables

    //Introducción a la Programación
    //Programa 7: EstSec7.psc
    //Solución de un sistema de ecuaciones lineales de dos variables
    //Programa que calcula las soluciones de un sistema de ecuaciones lineales de dos variables
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir CoeficienteX1, CoeficienteX2, CoeficienteY1, CoeficienteY2 Como Real;
    Definir ValorX, ValorY Como Real;
    Definir SolucionX, SolucionY Como Real;
```

```
    //Leer coeficientes y valores del sistema de ecuaciones lineales
    Escribir "Introduce el valor del coeficiente de x de la primera ecuación: ";
    Leer CoeficienteX1;
    Escribir "Introduce el valor del coeficiente de y de la primera ecuación: ";
    Leer CoeficienteY1;
    Escribir "Introduce la constante de la primera ecuación: ";
    Leer ValorX;
    Escribir "Introduce el valor del coeficiente de x de la segunda ecuación: ";
    Leer CoeficienteX2;
    Escribir "Introduce el valor del coeficiente de y de la segunda ecuación: ";
    Leer CoeficienteY2;
    Escribir "Introduce la constante de la segunda ecuación: ";
    Leer ValorY;

    //Realizar operaciones, cada cálculo (SolucionX y SolucionY)) es una línea o renglón
    SolucionX <- ((ValorX * CoeficienteY2) - (CoeficienteY1 * ValorY)) / ((CoeficienteX1 * CoeficienteY2) - (CoeficienteY1 *
    CoeficienteX2));
    SolucionY <- ((ValorY * CoeficienteX1) - (CoeficienteX2 * ValorX)) / ((CoeficienteX1 * CoeficienteY2) - (CoeficienteY1 *
    CoeficienteX2));

    //Mostrar las soluciones
    Escribir "El valor de X es: ", SolucionX;
    Escribir "El valor de Y es: ", SolucionY;

FinAlgoritmo
```

Ejercicio 8

Una persona desea invertir su capital en un banco y desea saber cuánto dinero ganará después de un mes si el banco le pagará intereses del 2% mensual, cree un algoritmo para solucionarlo.

```
Algoritmo GananciaInversion

    //Introducción a la Programación
    //Programa 8: EstSec8.psc
    //Inversión de capital
    //Programa que calcula cuánto dinero se ganará después de un mes
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Interes Como Real;
    Definir Dias, Capital, Ganancia Como Entero;

    //Leer Capital y Dias
    Escribir "Ingrese el monto a invertir: ";
    Leer Capital;
    Escribir "Ingrese el número total de días del mes a considerar: ";
    Leer Dias;

    //Asignación de tasa de interés
    Interes <- 0.02 / 30;

    //Realizar cálculo
    Ganancia <- Capital * Dias * Interes;

    //Mostrar resultado
    Escribir "La ganancia por cobrar después del mes es de: ";
```

```
    Escribir Ganancia;

FinAlgoritmo
```

Ejercicio 9

Un constructor sabe que necesita 0.5 metros cúbicos de arena por metro cuadrado de revoque a realizar. Escriba un algoritmo que le permita obtener la cantidad de arena necesaria para revocar una pared cualquiera, según sus medidas (largo y alto) expresadas en metros.

```
Algoritmo CantidadArena

    //Introducción a la Programación
    //Programa 9: EstSec9.psc
    //Cálculo de metros cúbicos de arena para revoque
    //Programa que calcula cuántos metros cúbicos de arena se necesitan para revoque
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Metros, Arena, Largo, Ancho Como Real;

    //Asignación de valor
    Metros <- 0.5;

    //Leer Largo y Ancho
    Escribir "Ingrese el largo de la pared en metros: ";
    Leer Largo;
    Escribir "Ingrese el ancho de la pared en metros: ";
    Leer Ancho;

    //Realizar cálculo
    Arena <- (Largo * Ancho) * Metros;

    //Mostrar resultado
    Escribir "La cantidad de arena necesaria es de: ";
    Escribir Arena;

FinAlgoritmo
```

Ejercicio 10

Construya el algoritmo tal que, dado el radio, la generatriz y la altura de un cono, calcule e imprima el área de la base, el área lateral, el área total y su volumen.

Consideraciones:
• El área de la base se calcula aplicando la siguiente fórmula: AreaBase = π * $Radio^2$
• El área lateral se calcula: AreaLateral = π * Radio * Generatriz
• El área total se calcula como: AreaTotal = π * Radio * (Generatriz + Radio)
• El volumen se calcula de la siguiente forma: Volumen = 1/3 * AreaBase * Altura

```
Algoritmo AreaCono

    //Introducción a la Programación
    //Programa 10: EstSec10.psc
    //Cálculo del área de la base, el área lateral, el área total y el volumen de un cono
    //Programa que calcula el área de la base, el área lateral, el área total y el volumen de un cono
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Radio, Generatriz, Altura, AreaBase, AreaLado, AreaTotal, Volumen Como Real;
```

```
    //Inicialización de variables
    Radio <- 0;
    Generatriz <- 0;
    Altura <- 0;
    AreaBase <- 0;
    AreaLado <- 0;
    Volumen <- 0;

    //Leer Radio, Generatriz y Altura
    Escribir "Ingrese el radio del cono: ";
    Leer Radio;
    Escribir "Ingrese la generatriz del cono: ";
    Leer Generatriz;
    Escribir "Ingrese la altura del cono: ";
    Leer Altura;

    //Realizar cálculos
    AreaBase <- PI * (Radio * Radio);
    AreaLado <- PI * Radio * Generatriz;
    AreaTotal <- PI * Radio * (Generatriz + Radio)
    Volumen <- (1 / 3) * AreaBase * Altura;

    //Mostrar resultados
    Escribir "El área de la base es: ";
    Escribir AreaBase;
    Escribir "El área lateral es: ";
    Escribir AreaLado;
    Escribir "El área total es: ";
    Escribir AreaTotal;
    Escribir "El volumen es: ";
    Escribir Volumen;

FinAlgoritmo
```

Ejercicio 11

Construya el algoritmo tal que, dado el radio de una esfera, calcule e imprima el área y su volumen

Consideraciones:
• El área de una esfera la calculamos de esta forma: Area = 4 * π * $Radio^2$
• El volumen como: Volumen = 4/3 * π * $Radio^3$

```
Algoritmo Esfera

    //Introducción a la Programación
    //Programa 11: EstSec11.psc
    //Cálculo del área y volumen de una esfera
    //Programa que calcula el área y el volumen de una esfera
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Area, Radio, Volumen, Prueba Como Real;

    //Leer Radio
    Escribir "Ingrese el valor del radio de la esfera: ";
    Leer Radio;

    //Realizar cálculos
    Area <- (4 * PI) * (Radio * Radio);
```

```
    Volumen <- (4 / 3)* PI * (Radio * Radio * Radio);

    //Mostrar resultados
    Escribir "El resultado del área de la esfera es: ";
    Escribir Area;
    Escribir "El resultado del Volumen de la esfera es:";
    Escribir Volumen;

FinAlgoritmo
```

Ejercicio 12

Construya el algoritmo tal que, dado como dato el lado de un hexaedro o cubo, calcule el área de la base, el área lateral, el área total y el volumen

Consideraciones:
• Para calcular el área de la base aplicamos la siguiente fórmula: AreaBase = $Lado^2$
• Para calcular el área lateral hacemos: AreaLateral = 4 * L^2
• Para calcular el área total hacemos: AreaTotal = 6 * L^2
• Para calcular el volumen hacemos: Volumen = L^3

```
Algoritmo Cubo

    //Introducción a la Programación
    //Programa 12: EstSec12.psc
    //Calcular el área de la base, el área lateral, el área total y el volumen de un hexaedro o cubo
    //Programa que calcula el área de la base, el área lateral, el área total y el volumen de un hexaedro o cubo
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Lado, AreaBase, AreaLado, AreaTotal, Volumen Como Real;

    //Leer Lado
    Escribir "Ingrese el valor del lado del Hexaedro o Cubo: ";
    Leer Lado;

    //Realizar cálculos
    AreaBase <- Lado * Lado;
    AreaLado <- 4 * (Lado * Lado);
    AreaTotal <- 6 * (Lado * Lado);
    Volumen <- Lado * Lado * Lado;

    //Mostrar resultados
    Escribir "El área de la base es: ";
    Escribir AreaBase;
    Escribir "El área lateral de la base es: ";
    Escribir AreaLado;
    Escribir "El área total de la base es: ";
    Escribir AreaTotal;
    Escribir "El volumen es: ";
    Escribir Volumen;

FinAlgoritmo
```

Ejercicio 13

Construya el algoritmo tal que, dadas las coordenadas de los puntos P1, P2 y P3 que corresponden a los vértices de un triángulo, calcule su perímetro.

Donde:

• X1 y Y1 representan las coordenadas en el eje de las X y las Y, del punto P1
• X2 y Y2 expresan las coordenadas en el eje de las X y las Y, del punto P2
• X3 y Y3 representan las coordenadas en el eje de las X y las Y, del punto P3

Consideraciones:
Para calcular la distancia entre dos puntos P1 y P2 hacemos:

$$Distancia = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

```
Algoritmo Triangulo

    //Introducción a la Programación
    //Programa 13: EstSec13.psc
    //Calcular perímetro de un triángulo dadas las coordenadas de los puntos de sus vértices
    //Programa que calcula perímetro de un triángulo dadas las coordenadas de los puntos de sus vértices
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir X1, X2, X3, Y1, Y2, Y3, Distancia1, Distancia2, Distancia3, Perimetro Como Real;

    //Leer variables
    Escribir "Introduce el valor de X1: ";
    Leer X1;
    Escribir "Introduce el valor de X2: ";
    Leer X2;
    Escribir "Introduce el valor de X3: ";
    Leer X3;
    Escribir "Introduce el valor de Y1: ";
    Leer Y1;
    Escribir "Introduce el valor de Y2: ";
    Leer Y2;
    Escribir "Introduce el valor de Y3: ";
    Leer Y3;

    //Realizar cálculos
    Distancia1 <- raiz((X1 - X2) ↑ 2 + (Y1 - Y2) ↑ 2)
    Distancia2 <- raiz((X2 - X3) ↑ 2 + (Y2 - Y3) ↑ 2)
    Distancia3 <- raiz((X3 - X1) ↑ 2 + (Y3 - Y1) ↑ 2)
    Perimetro <- Distancia1 + Distancia2 + Distancia3;

    //Mostrar resultado
    Escribir "El perímetro del triángulo es: ";
    Escribir Perimetro;

FinAlgoritmo
```

Ejercicio 14

Construya el algoritmo tal que, dadas las coordenadas de los puntos P1, P2 y P3 que corresponden a los vértices de un triángulo, calcule su superficie.

Donde:
• X1 y Y1 representan las coordenadas en el eje de las X y las Y, del punto P1.
• X2 y Y2 expresan las coordenadas en el eje de las X y las Y, del punto P2.
• X3 y Y3 representan las coordenadas en el eje de las X y las Y, del punto P3.

Consideraciones:
• Para calcular el área de un triángulo dadas de las coordenadas de los vértices que la componen, debemos aplicar la siguiente formula

$$Area = \frac{1}{2} * ¿X1 * (Y2 - Y3) + X2 * (Y3 - Y1) + X3 * (Y1 - Y2) \vee ¿$$

O bien, esta otra:

$$Area = \frac{1}{2} * ¿(X2 - X1) * (Y3 - Y1) - (X3 - X1) * (Y2 - Y1) \vee ¿$$

```
Algoritmo SuperficieTriangulo

    //Introducción a la Programación
    //Programa 14: EstSec14.psc
    //Calcular la superficie o área de un triángulo dadas las coordenadas de los puntos de sus vértices
    //Programa que calcula la superficie o área de un triángulo dadas las coordenadas de los puntos de sus vértices
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir X1, X2, X3, Y1, Y2, Y3, Area Como Real;

    //Leer variables
    Escribir "Introduce el valor de X1: ";
    Leer X1;
    Escribir "Introduce el valor de X2: ";
    Leer X2;
    Escribir "Introduce el valor de X3: ";
    Leer X3;
    Escribir "Introduce el valor de Y1: ";
    Leer Y1;
    Escribir "Introduce el valor de Y2: ";
    Leer Y2;
    Escribir "Introduce el valor de Y3: ";
    Leer Y3;

    //Realizar cálculo
    Area <- (((X1 * Y2) + (X2 * Y3) + (X3 * Y1)) – ((X1 * Y3) + (X3 * Y2) + (X2 * Y1))) / 2;

    //Mostrar resultado
    Escribir "El área del triángulo es: ";
    Escribir Area;

FinAlgoritmo
```

Ejercicio 15

Construya el algoritmo tal que, dado el perímetro de la base, la apotema y la altura de un prisma pentagonal; calcule el área de la base, el área lateral, el área total y el volumen

Consideraciones:

• Para calcular el área de la base, hacemos: $AreaBase = \frac{Perimetro * Apotema}{2}$

• Para calcular el área lateral, aplicamos la siguiente fórmula: AreaLateral=Perimetro * Altura
• Para calcular el área total hacemos: AreaTotal = 2 * AreaBase + AreaLateral
• Para calcular el volumen hacemos: Volumen = AreaBase * Altura

```
Algoritmo PrismaPentagonal

    //Introducción a la Programación
    //Programa 15: EstSec15.psc
    //Calcular el área de la base, el área lateral, el área total y el volumen de un prisma pentagonal
```

```
//Programa que calcula el área de la base, el área lateral, el área total y el volumen de un prisma pentagonal
//Autor: ® Fernando Jiménez Ávila © CCAP México

//Declarar variables
Definir AreaBase, Perimetro, Apotema, AreaLado, Altura, AreaTotal, Volumen Como Real;

//Leer variables
Escribir "Ingrese el perímetro de la base: ";
Leer Perimetro;
Escribir "Ingrese el valor de la apotema: ";
Leer Apotema;
Escribir "Ingrese el valor de la altura: ";
Leer Altura;

//Realizar cálculos
AreaBase <- (Perimetro * Apotema) / 2;
AreaLado <- Perimetro * Altura;
AreaTotal <- 2 * AreaBase + AreaLado;
Volumen <- AreaBase * Altura;

//Mostrar resultados
Escribir "El área de la base es: ";
Escribir AreaBase;
Escribir "El área lateral es: ";
Escribir AreaLado;
Escribir "El área total es: ";
Escribir AreaTotal;
Escribir "El volumen del prisma pentagonal es: ";
Escribir Volumen;

FinAlgoritmo
```

Ejercicio 16

Construya un algoritmo que calcule el monto total de una capital según sea el capital ingresado inicialmente y la tasa de interés vigente.

```
Algoritmo MontoCapital

//Introducción a la Programación
//Programa 17: EstSec17.psc
//calcular el monto total de una inversión
//Programa que calcula el monto total  de una inversión
//Autor: ® Fernando Jiménez Ávila © CCAP México

//Declarar variables
Definir Capital, Interes, Monto Como Real;

//Inicializar variables
Capital <- 0;
Interes <- 0;
Monto <- 0;

//Leer variables
Escribir "Ingrese el capital: ";
Leer Capital;
Escribir "Ingrese la tasa de interés: ";
Leer Interes;

//Realizar cálculo
Monto <- Capital * (1 + (Interes / 100));
```

```
        //Mostrar resultado
        Escribir "El monto es: ";
        Escribir Monto;

FinAlgoritmo
```

Ejercicio 17

Escribe un algoritmo que calcule e imprima los segundos que existen en el número de días ingresados por el usuario

```
Algoritmo SegundosPorDia

        //Introducción a la Programación
        //Programa 17: EstSec17.psc
        //Calcular número de segundos de los días solicitados
        //Programa que calcula el número de segundos  que existen en los días solicitados
        //Autor: ® Fernando Jiménez Ávila © CCAP México

        //Declarar variables
        Definir Dias, Segundos Como Real;

        //Leer Dias
        Escribir "Ingrese los días: ";
        Leer Dias;

        //Realizar cálculo
        Segundos <- Dias * 24 * 60 * 60;

        //Mostrar resultado
        Escribir "Los días: ";
        Escribir Dias;
        Escribir "Son equivalentes a estos segundos: ";
        Escribir Segundos;

FinAlgoritmo
```

Problemas con estructuras de selección

Ejemplo 1

```
Algoritmo AreadeCirculoCondicion

        Definir Radio, Area Como Real;

        Escribir "Introduce el radio del círculo: ";
        Leer Radio;

        Escribir "El radio del círculo es: ", Radio;

        Area = PI * (Radio * Radio);

        Si Area < 1000 Entonces

                Escribir "El área del círculo es ", Area;

        FinSi
```

```
FinAlgoritmo
```

Ejemplo 2

```
Algoritmo CalculoSueldoSemanal

	Definir Sueldo, Area Como Real;
	Definir HorasTrabajadas, HorasExtras Como Entero;

	Escribir "Introduce las horas trabajadas: ";
	Leer HorasTrabajadas;

	Escribir "Las horas trabajadas son: ", HorasTrabajadas;

	Si HorasTrabajadas <= 40 Entonces

		Sueldo <- HorasTrabajadas * 250;

	SiNo

		HorasExtras <- HorasTrabajadas - 40;
		Sueldo <- ((HorasTrabajadas - HorasExtras) * 250) + (HorasExtras * 125);

	FinSi

	Escribir "El sueldo total a pagar es: ", Sueldo;

FinAlgoritmo
```

Ejemplo 3

```
Algoritmo MenudeOperacionesBasicas

	Definir Numero1, Numero2, Resultado Como Real;
	Definir OperacionAritmetica Como Entero;

	Escribir "Introduce el primer número: ";
	Leer Numero1;
	Escribir "Introduce el segundo número";
	Leer Numero2;

	Escribir "..:::..Menú de operaciones aritméticas básicas..:::..";
	Escribir "1) Suma";
	Escribir "2) Resta";
	Escribir "3) Multiplicación";
	Escribir "4) División";
	Escribir "5) Cuadrado del primer número";
	Escribir "6) Raíz cuadrada del primer número";

	Escribir "Elige el número de la operación aritmética a realizar";
	Leer OperacionAritmetica;

	Segun OperacionAritmetica Hacer

		1:
			Resultado <- Numero1 + Numero2;
		2:
```

```
                Resultado <- Numero1 - Numero2;
        3:
                Resultado <- Numero1 * Numero2;
        4:
                Resultado <- Numero1 / Numero2;
        5:
                Resultado <- Numero1 * Numero1;
        6:
                Resultado <- raiz(Numero1);
        De Otro Modo:
                Escribir "No es una opción válida";

    Fin Segun

    Escribir "El resultado de la operación elegida es: ", Resultado;

FinAlgoritmo
```

Estructura selectiva simple

Ejercicio 1

Realizar un programa que indique si un número entero introducido por el usuario es positivo.

```
Algoritmo NumeroPositivo

    //Introducción a la Programación
    //Programa 1: EstSelSim1.psc
    //Determinar si un número es positivo.
    //Programa que determina o evalúa si un número introducido es positivo
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variable
    Definir Numero1 Como Real;

    //Leer Numero1
    Escribir "Introduce un número: ";
    Leer Numero1;

    //Evaluar si Numero1 es mayor que 0 (cero), es decir, si es un número positivo
    Si Numero1 > 0 Entonces

        //Si sí se cumple con la condición, mostrar mensaje
        Escribir "El número introducido es positivo";

    FinSi

        //Mostrar el número introducido
        Escribir "El número introducido es: ", Numero1;

FinAlgoritmo
```

Ejercicio 2

Escribir un programa en el que se indique si la calificación introducida es aprobatoria.

```
Algoritmo CalificacionAprobatoria

    //Introducción a la Programación
    //Programa 2: EstSelSim2.psc
    //Calificación aprobatoria.
    //Programa que determina si la calificación es aprobatoria
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variable
    Definir Calificacion Como Real;

    //Leer Calificacion
    Escribir "Introduce la calificación: ";
    Leer Calificacion;

    //Evaluar si Calificacion es mayor o igual que 6
    Si Calificacion >= 6 Entonces

        //Si sí se cumple con la condición, mostrar mensaje
        Escribir "La calificación es aprobatoria";

    FinSi

        //Mostrar Calificacion introducida
        Escribir "La calificación introducida es: ", Calificacion;

FinAlgoritmo
```

Ejercicio 3

Escribir un algoritmo que aplique un aumento del 15% al sueldo de un trabajador, si éste es menor a $1,000.00

```
Algoritmo SueldoTrabajador

    //Introducción a la Programación
    //Programa 3: EstSelSim3.psc
    //Calcular aumento de salario
    //Programa que calcula el aumento de salario cuando éste sea menor a $1,000.00
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Sueldo, NuevoSueldo Como Real;

    //Inicalizar variables
    Sueldo <- 0;
    NuevoSueldo <- 0;

    //Leer Sueldo
    Escribir “El sueldo del trabajador es: “;
    Leer Sueldo;

    //Evaluar si Sueldo es menor que $ 1.000.00
    Si (Sueldo < 1000) Entonces

        //Realizar cálculo
        NuevoSueldo <- (Sueldo * 1.15);

        //Mostrar resultado
```

```
            Escribir "El nuevo sueldo del trabajador es: ";
            Escribir NuevoSueldo;

        FinSi

FinAlgoritmo
```

Ejercicio 4

Escribe un algoritmo que calcule la temperatura de acuerdo con el número de sonidos emitidos por un grillo

```
Algoritmo SonidosGrillo

        //Introducción a la Programación
        //Programa 4: EstSelSim4.psc
        //Calcular temperatura de acuerdo con el número de sonidos emitidos por un grillo
        //Programa que calcula la temperatura de acuerdo con el número de sonidos que emite un grillo
        //Autor: ® Fernando Jiménez Ávila © CCAP México

        //Declarar variables
        Definir Numero1, Temperatura Como Real;

        //Inicializar variables
        Numero1 <- 0;
        Temperatura <- 0;

        //Leer Numero1
        Escribir "Ingrese el número de sonidos emitidos por el grillo: ";
        Leer Numero1;

        //Evaluar si Numero1 es mayor que 0 (cero)
        Si (Numero1 > 0) Entonces

            //Realizar cálculo
            Temperatura <- (Numero1 / 4) + 40;

            //Mostrar resultado
            Escribir "La temperatura es: ";
            Escribir Temperatura;

        FinSi

FinAlgoritmo
```

Estructura selectiva doble

Ejercicio 1

Escribir un algoritmo para lavar los platos de la comida.

```
Algoritmo LavarPlatos

        //Introducción a la Programación
        //Programa 1: EstSelDob1.psc
        //Lavar platos de la comida
        //Programa que muestra los pasos para lavar platos d ela comida
        //Autor: ® Fernando Jiménez Ávila © CCAP México

        //Declarar variables
```

```
Definir Traste, Estado Como Caracter;

//Leer Estado
Escribir "¿Cómo se encuentran los trastes?";
Escribir "¿Limpios o sucios?";
Leer Estado;

//Evaluar si Estado es igual a sucios o Sucios
Si Estado = "sucios" O Estado = "Sucios" Entonces

        //Si se cumple con la condición, mostrar mensajes
        Escribir "Lavar trastes";
        Escribir "Mezclar jabón con agua";
        Escribir "Meter esponja en la mezcla";
        Escribir "Fregar trastes con la esponja";
        Escribir "Enjuagar trastes";
        Escribir "Secar trastes";
        Escribir "Guardar trastes";

SiNo

        //Si no se cumple con la condición, mostrar mensaje
        Escribir "No lavarlos";

FinSi

FinAlgoritmo
```

Ejercicio 2

Escribir un algoritmo para reparar un pinchazo de bicicleta.

```
Algoritmo Pinchazo

        //Introducción a la Programación
        //Programa 2: EstSelDob2.psc
        //Reparación de un pinchazo de una bicicleta
        //Programa que muestra los pasos para reparar el pinchazo de una bicicleta
        //Autor: ® Fernando Jiménez Ávila © CCAP México

        //Declarar variable
        Definir Estado Como Entero;

        //Leer Estado
        Escribir "¿Las llantas de la bicicleta se encuentran ponchadas?";
        Escribir "1.Si 2.No";
        Leer Estado;

        //Evaluar si Estado es igual a 1
        Si (Estado = 1) Entonces

                //Si se cumple con la condición, mostrar mensajes
                Escribir "Desmontar la rueda";
                Escribir "Quitar la cubierta";
                Escribir "Sacar la cámara";
                Escribir "Inflar la cámara";
                Escribir "Meter la pieza en un cubo de agua";
                Escribir "Checar de dónde salen las burbujas";
                Escribir "Marcar el pinchazo";
```

```
            Escribir "Aplicar pegamento en el área dañada";
            Escribir "Esperar que seque el pegamento";
            Escribir "Parchar el pinchazo";
            Escribir "Aplicar presión en el pinchazo para que quede bien el parche";
            Escribir "Montar la cámara";
            Escribir "Montar la cubierta";
            Escribir "Montar la llanta";
            Escribir "Inflar la llanta";

        SiNo

            //Si no se cumple con la condición, mostrar mensaje
            Escribir "No hay que parchar la llanta";

        FinSi

FinAlgoritmo
```

Ejercicio 3

Hacer un programa que calcule el pago de horas extras.

```
Algoritmo PagaHorasExtras

    //Introducción a la Programación
    //Programa 3: EstSelDob3.psc
    //Pago de horas extras.
    //Programa que calcula la paga por horas trabajadas, si rebasa de 40 horas, se paga una tasa adicional por hora extra
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Tasa, HorasTrabajadas, Paga, Salario Como Real;

    //Asignación de variable
    Salario <- 200;

    //Leer variables
    Escribir "Introduce la tasa a pagar: (Introducir la tasa como entero)";
    Leer Tasa;
    Escribir "Introduce las horas trabajadas: ";
    Leer HorasTrabajadas;

    //Evaluar si HorasTrabajadas es menor o igual a 40
    Si HorasTrabajadas <= 40 Entonces

        //Si se cumple con la condición, realizar cálculo
        Paga <- Salario * 40;

    SiNo

        //Si no se cumple con la condición, realizar cálculo
        Paga <- (Salario * 40) + ((HorasTrabajadas - 40) * (Tasa / 100));

    FinSi

    //Mostrar resultado
    Escribir "La paga total a hacer es de: ", Paga;
```

```
FinAlgoritmo
```

Ejercicio 4

Construya un algoritmo dado como dato la calificación de un alumno en un examen, escriba “APROBADO” si su calificación es mayor o igual que 8 y “REPROBADO” en caso contrario

```
Algoritmo Calificacion

    //Introducción a la Programación
    //Programa 4: EstSelDob4.psc
    //Determinar si el alumno aprueba o no aprueba
    //Programa que determina si un alumno aprueba o no aprueba
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variable
    Definir Calificacion Como Real;

    //Inicializar variable
    Calificacion <- 0;

    //Leer Calificacion
    Escribir “Ingrese la calificación del alumno: “;
    Leer Calificacion;

    //Evaluar si Calificacion es mayor o igual a 8
    Si (Calificacion >= 8) Entonces

        //Mostrar resultado si sí se cumple con la condición
        Escribir “El alumno está APROBADO “;

    SiNo

        //Mostrar resultado si no se cumple con la condición
        Escribir “El alumno está REPROBADO “;

    FinSi

FinAlgoritmo
```

Ejercicio 5

Construya un algoritmo dado como dato el sueldo de un trabajador, aplique un aumento del 15% si su sueldo es inferior a $ 1,000.00 y el 12% en caso contrario, imprima el nuevo sueldo del trabajador

```
Algoritmo AumentoSueldo

    //Introducción a la Programación
    //Programa 5: EstSelDob5.psc
    //Determinar aumento de salario
    //Programa que determina el aumento de salario
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declaración de variables
    Definir Sueldo, NuevoSueldo Como Real;

    //Inicialización de variables
    Sueldo <- 0;
    NuevoSueldo <- 0;
```

```
    //Leer Sueldo
    Escribir "Ingrese el sueldo del trabajador: ";
    Leer Sueldo;

    //Evaluar si Sueldo es menor que $ 1,000.00
    Si (Sueldo < 1000) Entonces

        //Si se cumple con la condición realizar cálculo
        NuevoSueldo <- Sueldo * 1.5;

        //Mostrar resultado
        Escribir "El nuevo sueldo del trabajador es: ";
        Escribir NuevoSueldo;

    SiNo

        //Si no se cumple con la condición, realizar el cálculo
        NuevoSueldo <- Sueldo * 1.2;

        //Mostrar resultado
        Escribir "El nuevo sueldo del trabajador es: ";
        Escribir NuevoSueldo;

    FinSi

FinAlgoritmo
```

Ejercicio 6

Construye un algoritmo dado como datos la matrícula y 5 calificaciones de un alumno; imprima la matrícula, el promedio y la palabra "Aprobado" si el alumno tiene un promedio mayor o igual que 6, y la palabra "No aprobado", en caso contrario.

```
Algoritmo Promedio

    //Introducción a la Programación
    //Programa 6: EstSelDob6.psc
    //Determinar si el alumno aprueba o no aprueba con base en el promedio
    //Programa que determina si se aprueba o no se aprueba con base en el promedio
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Nota1, Nota2, Nota3, Nota4, Nota5, Promedio1 Como Real;
    Definir Matricula Como Cadena;

    //Inicializar variables
    Nota1 <- 0;
    Nota2 <- 0;
    Nota3 <- 0;
    Nota4 <- 0;
    Nota5 <- 0;
    Promedio1 <- 0;
    Matricula <- "";

    //Leer variables
    Escribir "Ingrese la matricula: ";
    Leer Matricula;
    Escribir "Ingrese las calificaciones: ";
    Leer Nota1, Nota2, Nota3, Nota4, Nota5;
```

```
    //Realizar cálculos
    Promedio1 <- (Nota1 + Nota2 + Nota3 + Nota4 + Nota5) / 5

    //Mostrar resultados
    Escribir "Matricula: ";
    Escribir Matricula;
    Escribir "Promedio: ";
    Escribir Promedio1;

    //Evaluar si el promedio es igual o mayor que 6
    Si (Promedio1 >= 6) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "El alumno está APROBADO ";

    SiNo

        //Si no se cumple con la condición, mostrar mensaje
        Escribir "El alumno está REPROBADO ";

    FinSi

FinAlgoritmo
```

Ejercicio 7

Construye un algoritmo que, dados el nombre del grupo y el número de alumnos preinscritos, exprese el nombre del grupo, el número de alumnos inscritos y si el grupo será abierto o cerrado, puesto que, para abrir un grupo, se necesitan mínimo 30 alumnos

```
Algoritmo Grupo

    //Introducción a la Programación
    //Programa 7: EstSelDob7.psc
    //Determinar si un grupo es cerrado o no
    //Programa que determina si un grupo es cerrado, cupo lleno, o es abierto
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Alumnos Como Entero;
    Definir Nombre, Grupo Como Caracter;

    //Inicializar variables
    Alumnos <- 0;
    Nombre <- "";
    Grupo <- "";

    //Leer variables
    Escribir "Ingrese el nombre del grupo: ";
    Leer Nombre;
    Escribir "Ingrese el número de alumnos preinscritos: ";
    Leer Alumnos;
    Escribir "Grupo: ";
    Escribir Nombre;

    //Evaluar el número de alumnos
    Si (Alumnos >= 30) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "Grupo CERRADO";
```

```
        SiNo

            //Si no se cumple con la condición, mostrar mensaje
            Escribir "Grupo ABIERTO";

        FinSi

FinAlgoritmo
```

Ejercicio 8

Escribe un algoritmo que calcule el descuento considerando las siguientes especificaciones:

1.- Si el monto es mayor a $100, se aplica un descuento del 10% sobre la compra.
2.- Si el monto es menor a $100, se aplica un descuento del 2% sobre el monto total de la compra.

```
Algoritmo Descuento

        //Introducción a la Programación
        //Programa 8: EstSelDob8.psc
        //Calcular descuentos
        //Programa que determina la aplicación de descuentos
        //Autor: ® Fernando Jiménez Ávila © CCAP México

        //Declarar variables
        Definir Monto, Descuento, MontoDescuento Como Real;

        //Inicializar variables
        Monto <- 0;
        Descuento <- 0;
        MontoDescuento <- 0;

        //Leer Monto
        Escribir "Ingrese el monto: ";
        Leer Monto;

        //Evaluar el Monto
        Si (Monto >= 100) Entonces

            //Si se cumple con la condición, realizar cálculos
            Descuento <- Monto * 0.1;
            MontoDescuento <- Monto – Descuento;

        SiNo

            //Si no se cumple con la condición, realizar cálculos
            Descuento <- Monto * 0.02;
            MontoDescuento <- Monto – Descuento;

        FinSi

        //Mostrar resultados
        Escribir "El descuento es de: ";
        Escribir Descuento;
        Escribir "El monto con descuento es: ";
        Escribir MontoDescuento;

FinAlgoritmo
```

Ejercicio 9

Escribe un algoritmo que solicite el nombre del jugador, número de goles; y que al jugador que tenga 6 goles o más, imprima en pantalla "GOLEADOR", en caso contrario, deberá imprimirse "NO GOLEADOR

```
Algoritmo GolesAnotados

    //Introducción a la Programación
    //Programa 9: EstSelDob9.psc
    //Determinar si un jugador es goleador o no es goleador
    //Programa que determina si un jugador es goleador o no es goleador
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Goles Como Entero;
    Definir Nombre Como Caracter;

    //Inicializar variables
    Goles <- 0;
    Nombre <- "";

    //Leer variables
    Escribir "Ingrese el nombre del jugador: ";
    Leer Nombre;
    Escribir "Ingrese la cantidad de goles anotados: ";
    Leer Goles;

    //Mostrar nombre del jugador
    Escribir "El jugador: ";
    Escribir Nombre;

    //Evaluar goles
    Si (Goles >= 6) Entonces

        //Si sí se cumple con la condición, mostrar mensaje
        Escribir "Es GOLEADOR";

    SiNo

        //Si no se cumple con la condición, mostrar mensaje
        Escribir "Es NO GOLEADOR";

    FinSi

FinAlgoritmo
```

Ejercicio 10

Escribe un algoritmo que identifique si el número ingresado es par o impar

```
Algoritmo ParImpar

    //Introducción a la Programación
    //Programa 10: EstSelDob10.psc
    //Determinar si un número es par o es impar
    //Programa que determina si un número es par o es impar
    //Autor: ® Fernando Jiménez Ávila © CCAP México
```

```
    //Declarar variable
    Definir Numero1 Como Real;

    //Inicializar variable
    Numero1 <- 0;

    //Leer Numero1
    Escribir "Ingrese el número: ";
    Leer Numero1;

    //Evaluar si el resto de Numero1 con 2 es 0 (cero)
    Si (Numero1 MOD 2 = 0) Entonces

        //Si sí se cumple con la condición, mostrar mensaje
        Escribir "El número es PAR ";

    SiNo

        //Si no se cumple con la  condición, mostrar mensaje
        Escribir "El número es IMPAR";

    FinSi

FinAlgoritmo
```

Ejercicio 11

Escribe un algoritmo que identifique si un producto va a ser pagado en efectivo o mediante tarjeta, los aspectos a considerar son:

1.- Si el costo del producto a pagar es mayor a $200, se paga en efectivo
1.- Si el costo del producto es menor a $200, se paga con tarjeta

```
Algoritmo CostoProducto

    //Introducción a la Programación
    //Programa 11: EstSelDob11.psc
    //Determinar si se paga con efectivo o con tarjeta
    //Programa que determina si se paga con efectivo o se paga con tarjeta
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Precio Como Real;
    Definir Nombre Como Caracter;

    //Inicializar variables
    Precio <- 0;
    Nombre <- "";

    //Leer variables
    Escribir "Ingrese el nombre del producto: ";
    Leer Nombre;
    Escribir "Ingrese el precio del producto: ";
    Leer Precio;

    //Evaluar si Precio es mayor o igual a 200
    Si (Precio >= 200) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "El producto se paga en efectivo";
```

```
    SiNo

        //Si no se cumple con la condición, mostrar mensaje
        Escribir "El producto se paga con tarjeta";

    FinSi

FinAlgoritmo
```

Ejercicio 12

Construye un algoritmo, dado como datos de entrada tres números enteros. Determine si los mismos están en orden creciente.

```
Algoritmo NumerosCrecientes

    //Introducción a la Programación
    //Programa 12: EstSelDob12.psc
    //Determinar si tres números están en orden creciente
    //Programa que determina si tres números introducidos están en orden creciente
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Numero1, Numero2, Numero3 Como Real;

    //Inicializar variables
    Numero1 <- 0;
    Numero2 <- 0;
    Numero3 <- 0;

    //Leer variables
    Escribir "Ingrese el valor del primer número: ";
    Leer Numero1;
    Escribir "Ingrese el valor del segundo número: ";
    Leer Numero2;
    Escribir "Ingrese el valor del tercer número: ";
    Leer Numero3;
    Escribir "Los números ingresados son: ";
    Escribir Numero1,", ", Numero2, ", ", Numero3,".";

    //Evaluar el orden de los números
    Si (Numero2 > Numero1) Y (Numero2 > Numero3) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "Los números están en orden creciente";

    SiNo

        //Si no se cumple con la condición, mostrar mensaje
        Escribir "Los números no están en orden creciente";

    FinSi

FinAlgoritmo
```

Estructuras selectivas anidadas

Ejercicio 1

Escribe un algoritmo que imprima el nombre y marcador con el cual es ganador un equipo en cierto partido de futbol, se debe solicitar el nombre de los equipos del partido, y la cantidad de goles anotado por cada uno, se deben considerar también el número de goles anotados en los penaltis del partido.

```
Algoritmo PartidoFinal

	//Introducción a la Programación
	//Programa 1: EstSelAni1.psc
	//Determinar equipo ganador
	//Programa que determina el equipo ganador
	//Autor: ® Fernando Jiménez Ávila © CCAP México

	//Declarar variables
	Definir GolesA, GolesB, PenalesA, PenalesB Como Entero;
	Definir EquipoA, EquipoB Como Caracter;

	//Inicializarr variables
	GolesA <- 0;
	GolesB <- 0;
	PenalesA <- 0;
	PenalesB <- 0;
	EquipoA <- “”;
	EquipoB <- “”;

	//Leer variables
	Escribir “Ingrese el nombre del equipo A: “;
	Leer EquipoA;
	Escribir “Ingrese los goles del equipo A: “;
	Leer GolesA;
	Escribir “Ingrese el nombre del equipo B: “;
	Leer EquipoB;
	Escribir “Ingrese los goles del equipo B: “;
	Leer GolesB;

	//Evaluar si GolesA es igual a GolesB
	Si (GolesA = GolesB) Entonces

		//Si se cumple con la condición, leer variables
		Escribir “Ingrese los penales anotados del equipo A: “;
		Leer PenalesA;
		Escribir “Ingrese los penales anotados del equipo B: “;
		Leer PenalesB;

		//Evaluar si PenalesA es mayor que PenalesB
		Si (PenalesA > PenalesB) Entonces

			//Si se cumple con la condición, mostrar mesnaje
			Escribir “Ganó el equipo “;
			Escribir EquipoA;

		SiNo

			//Si no se cumple con la condición, mostrar mensaj
			Escribir “Ganó el equipo “;
			Escribir EquipoB;

		FinSi

	FinSi
```

```
    //Evaluar si GolesA es mayor que GolesB
    Si (GolesA > GolesB) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "Ganó el equipo ";
        Escribir EquipoA;

    SiNo

        //Si no se cumple con la condición, mostrar mensaje
        Escribir "Ganó el equipo ";
        Escribir EquipoB;

    FinSi

FinAlgoritmo
```

Ejercicio 2

Construye un algoritmo dado como dato un número entero. Determine e imprima si el mismo es positivo, negativo o nulo, es decir, cero.

```
Algoritmo Numeros

    //Introducción a la Programación
    //Programa 2: EstSelAni2.psc
    //Determinar si un número es positivo, negtivo o nulo
    //Programa que determina si un número introducido es positivo, negativo o nulo
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variable
    Definir Numero1 Como Entero;

    //Inicializar variable
    Numero1 <- 0;

    //Leer variable
    Escribir "Ingrese el número entero: ";
    Leer Numero1;

    //Evaluar si Numero1 es igual a 0 (cero)
    Si (Numero1 = 0) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "El número es NULO ";

    SiNo

        //Si no se cumple con la condición, evaluar si Numero1 es igual o mayor que 1
        Si (Numero1 >= 1) Entonces

            //Si se cumple con la condición, mostrar mensaje
            Escribir "El número es POSITIVO";

        SiNo

            //Si no se cumple con la condición, evaluar si Numero1 es menor que 0 (cero)
            Si (Numero1 < 0) Entonces

                //Si se cumple con la condición, mostrar mensaje
```

```
                    Escribir "El número es NEGATIVO";

                FinSi

            FinSi

        FinSi

FinAlgoritmo
```

Ejercicio 3

Construya un algoritmo dado el monto de la compra de un cliente, que determine lo que se debe pagar.
En una tienda efectúan un descuento a los clientes dependiendo del monto de la compra. El descuento se efectúa con base en el siguiente criterio:

1. Si el monto es menor que $500.00, no hay descuento.
2. Si el monto está comprendido entre $500 y $1000.00, tiene un 5% de descuento.
3. Si el monto está comprendido entre $1000 y $7000.00, tiene un descuento del 11%.
4. Si el monto está comprendido entre $7,000 y $15,000.00, tiene un 18% de descuento.
5. Si el monto es mayor a $15,000.00, tiene un 25% de descuento.

```
Algoritmo MontoDescuento

    //Introducción a la Programación
    //Programa 3: EstSelAni3.psc
    //Determinar descuentos
    //Programa que determina descuentos con base al monto de la compra
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Monto, Descuento Como Real;

    //Inicializar variables
    Monto <- 0;
    Descuento <- 0;

    //Leer Monto
    Escribir "Ingrese el monto de la compra: ";
    Leer Monto;

    //Evaluar si Monto es menor o igual 500
    Si (Monto <= 500) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "El monto no tiene descuento";
        Escribir "El cliente debe pagar";
        Escribir Monto;

    SiNo

        //Si no se cumple con la condición, evaluar si Monto es mayor a 500 y menor a 1,000
        Si (Monto > 500) Y (Monto <= 1000) Entonces

            //Si se cumple con la condición, realizar cálculo
            Descuento <- Monto - (Monto * 0.05);

            //Mostrar resultado
            Escribir "El cliente debe pagar";
            Escribir Descuento;
```

```
        SiNo

            //Si no se cumple con la condición, evaluar si Monto es mayor a 1,000 y menor o igual a 7,000
            Si (Monto > 1000) Y (Monto <= 7000) Entonces

                //Si se cumple con la condición, realizar cálculo
                Descuento <- Monto – (Monto * 0.11);

                //Mostrar resultado
                Escribir "El cliente debe pagar";
                Escribir Descuento;

            SiNo

                //Si no se cumple con la condición, evaluar si Monto es mayor a 7,000 y menos o igual a 15,000
                Si (Monto > 7000) Y (Monto <= 15000) Entonces

                    //Si se cumple con la condición, realizar cálculo
                    Descuento <- Monto – (Monto * 0.18);

                    //Mostrar resultado
                    Escribir "El cliente debe pagar";
                    Escribir Descuento;

                SiNo

                    //Si no se cumple con la condición, evaluar si Monto es mayor que 15,000
                    Si (Monto > 15000) Entonces

                        //Si se cumple con la condición, realizar cálculo
                        Descuento <- Monto – (Monto * 0.25);

                        //Mostrar resultado
                        Escribir "El cliente debe pagar";
                        Escribir Descuento;

                    FinSi

                FinSi

            FinSi

        FinSi

FinAlgoritmo
```

Ejercicio 4

Diseñe un algoritmo que lea el costo básico de un artículo y calcule su precio total (precio total igual a precio básico más impuesto)

```
Algoritmo Impuesto

    //Introducción a la Programación
    //Programa 4: EstSelAni4.psc
    //Calcular precio total
    //Programa que calcula el precio total de un artículo
```

```
//Autor: ® Fernando Jiménez Ávila © CCAP México

//Declarar variables
Definir Precio, Impuesto, PrecioTotal Como Real;

//Inicializar variables
Precio <- 0;
Impuesto <- 0;
PrecioTotal <- 0;

//Leer Precio
Escribir "Ingrese el precio del producto: ";
Leer Precio;

//Evaluar si Precio es menor que 20
Si (Precio < 20) Entonces

        //Si se cumple la condición, mostrar mensaje
        Escribir "El producto no paga impuestos";

        //Asignar Precio a PrecioTotal
        PrecioTotal <- Precio;

SiNo
        //Si no se cumple con la condición, evaluar si Precio es mayor o igual a 20 y es menor o igual a 40
        Si (Precio >= 20) Y (Precio <= 40) Entonces

                //Si se cumple con la condición, realizar cálculo
                Impuesto <- Precio * 1.30;

        SiNo

                //Si no se cumple con la condición, evaluar si Precio es mayor a 40 y menor o igual a 500
                Si (Precio > 40) Y (Precio <= 500) Entonces

                        //Si se cumple con la condición, realizar cálculo
                        Impuesto <- Precio * 1.40;

                SiNo

                        //Si no se cumple con la condición, realizar cálculo, lo que significa que el Precio es mayor de 500
                        Impuesto <- Precio * 1.50;

                FinSi

        FinSi

FinSi

//Mostrar resultados
Escribir "El precio del producto es: ";
Escribir Precio;
Escribir "El impuesto del producto es: ";
Escribir Impuesto;
Escribir "El precio total del producto es: ";
Escribir Precio + Impuesto;

FinAlgoritmo
```

Ejercicio 5

Construya un algoritmo que, dados la matrícula de un alumno, la carrera en la que está inscrito, su semestre y su promedio; determine si el mismo es apto para pertenecer a algunas de las facultades menores que tiene la universidad.

```
Algoritmo AlumnoAceptado

    //Introducción a la Programación
    //Programa 5: EstSelAni5.psc
    //Determinar la pertenencia a alguna facultad
    //Programa que determina si puede pertenecer a alguna facultad
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Cambio, Semestre, Promedio1 Como Real;
    Definir Carrera, Matricula Como Caracter;

    //Leer variables
    Escribir "¿En qué carrera está inscrito?";
    Leer Carrera;
    Escribir "Elige alguna de las carreras:";
    Escribir "Economía (1)";
    Escribir "Computación (2)";
    Escribir "Administración (3)";
    Escribir "Contabilidad (4)";
    Escribir "¿A qué carrera te quieres cambiar?";
    Leer Cambio;
    Escribir "Ingrese la matricula del alumno";
    Leer Matricula;
    Escribir "¿En qué semestre está inscrito?";
    Leer Semestre;
    Escribir "¿Cuál es su promedio?";
    Leer Promedio1;

    //Evaluar Cambio
    Si (Cambio = 1) Entonces

        //Si se cumple con la condición, evaluar Semestre y Promedio1
        Si (Semestre >= 6) Y (Promedio1 >= 8.8) Entonces

            //Si se cumple con la condición, mostrar mensaje
            Escribir "APROBADO";

        SiNo

            //Si no se cumple con la condición, mostrar mensaje
            Escribir "NO APROBADO";

        FinSi

    FinSi

    //Evaluar Cambio
    Si (Cambio = 2) Entonces

        //Si se cumple con la condición, evaluar Semestre y Promedio1
        Si (Semestre >= 6) Y (Promedio1 >= 8.5) Entonces

            //Si se cumple con la condición, mostrar mensaje
            Escribir "APROBADO";
```

```
            SiNo
                //Si no se cumple con la condición, mostrar mensaje
                Escribir "NO APROBADO";
            FinSi
        FinSi
        //Evaluar cambio
        Si (Cambio = 3) Entonces
            //Si se cumple con la condición, evaluar Semestre y Promedio1
            Si (Semestre >= 5) Y (Promedio1 >= 8.5) Entonces
                //Si se cumple con la condición, mostrar mensaje
                Escribir "APROBADO";
            SiNo
                //Si no se cumple con la condición, mostrar mensaje
                Escribir "NO APROBADO";
            FinSi
        FinSi
        //Evaluar Cambio
        Si (Cambio = 4) Entonces
            //Si se cumple con la condición, evaluar Semestre y Promedio1
            Si (Semestre >= 5) Y (Promedio1 >= 8.5) Entonces
                //Si se cumple con la condición, mostrar mensaje
                Escribir "APROBADO";
            SiNo
                //Si no se cumple con la condición, mostrar mensaje
                Escribir "NO APROBADO";
            FinSi
        FinSi
FinAlgoritmo
```

Ejercicio 6

Calcule el aumento de sueldo Para un grupo de empleados de una empresa, teniendo en cuenta el siguiente criterio:

1. Si el sueldo es inferior a $1,000.00, aumento del 15%
2. Si el sueldo es mayor o igual a $1,000.00, aumento del 12 %

Imprima el sueldo nuevo del trabajador y el total de la nómina de la empresa considerando este nuevo aumento.

```
Algoritmo AumentoSueldo
    //Introducción a la Programación
    //Programa 6: EstSelAni6.psc
```

```
//Cálculo de aumento y total de la nómina
//Programa que calcula el aumento de salario y el total de la nómina con el aumento
//Autor: ® Fernando Jiménez Ávila © CCAP México

//Declarar variables
Definir Trabajadores, Sueldo, SueldoNuevo, Nomina Como Real;

//Inicializar variables
Trabajadores <- 0;
Sueldo <- 0;
SueldoNuevo <- 0;
Nomina <- 0;

//Leer variables
Escribir "¿Cuántos trabajadores tiene el grupo?";
Leer Trabajadores;
Escribir "¿Cuál es el sueldo actual?";
Leer Sueldo;

//Evaluar si Sueldo es menor que $ 1,000.00
Si (Sueldo < 1000) Entonces

        //Si se cumple con la condición, realizar cálculo
        SueldoNuevo <- Sueldo * 1.15;

SiNo

        //Si no se cumple con la condición, evaluar si Sueldo es mayor o igual a $ 1,000.00
        Si (Sueldo >= 1000) Entonces

                //Si se cumple con la condición, realizar cálculo
                SueldoNuevo <- Sueldo * 1.12;

        FinSi

FinSi

//Obtener el total de la nómina
Nomina <- SueldoNuevo * Trabajadores;

//Mostrar resultados
Escribir "El nuevo sueldo del grupo de trabajadores será de: ";
Escribir SueldoNuevo;
Escribir "El total de nómina es: ";
Escribir Nomina;

FinAlgoritmo
```

Estructura selectiva múltiple

Ejercicio 1

Escribir un algoritmo dado como datos: Numero, Valor y Resultado obteniendo los resultados según las siguientes funciones:

Opción	Resultado
1	100 * Valor
2	100 ^ Valor
3	100 / Valor

```
Algoritmo Funcion

    //Introducción a la Programación
    //Programa 1: EstSelMul1.psc
    //Obtener resultados según funciones
    //Programa que determina resultados según funciones
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Opcion, Valor Como Entero;
    Definir Resultado Como Real;

    //Inicializar variables
    Opcion <- 0;
    Valor <- 0;
    Resultado <- 0;

    //Leer variables
    Escribir "Ingrese la opción a ejecutar: ";
    Escribir "(1) Multiplicar el número * 100,";
    Escribir "(2) Elevar como potencia el número a 100, ";
    Escribir "(3) Dividir el valor / 100.";
    Leer Opcion;
    Escribir "Ingrese el valor a trabajar";
    Leer Valor;

    //Evaluar Opcion según sea la condición
    Segun (Opcion) Hacer

        1:
        Resultado <- 100 * Valor;
        2:
        Resultado <- 100 ↑ Valor;
        3:
        Resultado <- 100 / Valor;

    De Otro Modo:

        Resultado <- 0;

    FinSegun

    //Mostrar resultado
    Escribir "El resultado de la función es:";
    Escribir Resultado;

FinAlgoritmo
```

Ejercicio 2

Construye un algoritmo dado como datos la categoría y el sueldo de un trabajador, calcule el aumento correspondiente, teniendo en cuenta la siguiente tabla:

Categoría	Porcentaje
1	15%
2	10%
3	8%
4	7%

```
Algoritmo SueldoCategoria

    //Introducción a la Programación
    //Programa 2: EstSelMul2.psc
    //Calcular aumento de salario de acuerdo a la tabla
    //Programa que calcula el aumento de salario de acuerdo con la categoría
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Categoria Como Entero;
    Definir Sueldo, NuevoSueldo Como Real;

    //Inicializar variables
    Categoria <- 0;
    Sueldo <- 0;
    NuevoSueldo <- 0;

    Escribir "Ingrese la categoría del trabajador: ";
    Leer Categoria;
    Escribir "Ingrese el sueldo del trabajador:";
    Leer Sueldo;

    //Evaluar Categoria según sea la condición
    Segun (Categoria) Hacer

        1:
        NuevoSueldo <- Sueldo * 1.15;
        2:
        NuevoSueldo <- Sueldo * 1.10;
        3:
        NuevoSueldo <- Sueldo * 1.08;
        4:
        NuevoSueldo <- Sueldo * 1.07;

    De Otro Modo:

        NuevoSueldo <- Sueldo;

    FinSegun

    //Mostrar resultados
    Escribir "La categoría del trabajador es:";
    Escribir Categoria;
    Escribir "El nuevo sueldo del trabajador es:";
    Escribir NuevoSueldo;

FinAlgoritmo
```

Ejercicio 3

El costo de las llamadas telefónicas internacionales depende de la zona geográfica en la que se encuentra el país de destino y el número de minutos hablados, en la siguiente tabla se presenta el costo de un minuto por zona; a cada zona se le asignado una clave. Construye un algoritmo que permita calcular e imprimir el costo total de una llamada:

Clave	Zona	Precio
12	América Norte	2
15	América Central	2.2
18	América Sur	4.5
19	Europa	3.5

23	Asia	6
25	África	6
29	Oceanía	5

```
Algoritmo CostoLlamada

    //Introducción a la Programación
    //Programa 1: EstSelMul1.psc
    //Calcular costo de llamada
    //Programa que calcul el costo de una llamada
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Costo, Zona, Minutos Como Real;

    //Inicializar variables
    Costo <- 0;
    Zona <- 0;
    Minutos <- 0;

    //Leer variables
    Escribir "Claves de las zonas geográficas:";
    Escribir "(12) América Norte";
    Escribir "(15) América Central";
    Escribir "(18) América Sur";
    Escribir "(19) Europa";
    Escribir "(23) Asia";
    Escribir "(25) África";
    Escribir "(29) Oceanía";
    Escribir "Ingrese la clave de la zona: ";
    Leer Zona;
    Escribir "Ingrese los minutos: ";
    Leer Minutos;

    //Evaluar Zona según sea la condición
    Según (Zona) Hacer

        12:
        costo <- Minutos * 2;
        15:
        costo <- Minutos * 2.2;
        18:
        costo <- Minutos * 4.5;
        19:
        costo <- Minutos * 3.5;
        23:
        costo <- Minutos * 6;
        25:
        costo <- Minutos * 6;
        29:
        costo <- Minutos * 5;

    De Otro Modo:

        Escribir "No es una clave válida";
        Costo <- Minutos;

    FinSegun

    //Mostrar resultado
```

```
    Escribir "El costo total de la llamada es de: ";
    Escribir Costo;

FinAlgoritmo
```

Ejercicio 4

Escriba el algoritmo que permita calcular lo que hay que pagarle a un trabajador teniendo en cuenta su sueldo y las horas extras trabajadas para el pago de horas extras se toma en cuenta la categoría del trabajador.

Categoría	Precio Hora Extra
1	30
2	38
3	50
4	70

```
Algoritmo HorasExtras

    //Introducción a la Programación
    //Programa 1: EstSelMul1.psc
    //Cálculo de horas extras
    //Programa que calcula horas extras según la categoría
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Sueldo, HorasExtras, NuevoSueldo Como Real;
    Definir Categoria Como Entero;

    //Inicializar variables
    Categoria <- 0;
    Sueldo <- 0;
    HorasExtras <- 0;
    NuevoSueldo <- 0;

    //Leer variables
    Escribir "Categorías";
    Escribir "(1) (2) (3) (4)";
    Escribir "Ingrese la categoría del trabajador: ";
    Leer Categoria;
    Escribir "Ingrese el sueldo del trabajador";
    Leer Sueldo;
    Escribir "Ingrese las horas extras del trabajador";
    Leer Sueldo;

    //Evaluar categoría, según sea la condición
    Segun (Categoria) Hacer

        1:
        NuevoSueldo <- Sueldo + (HorasExtras * 30);
        2:
        NuevoSueldo <- Sueldo + (HorasExtras * 38);
        3:
        NuevoSueldo <- Sueldo + (HorasExtras * 50);
        4:
        NuevoSueldo <- Sueldo + (HorasExtras * 70);

    De Otro Modo:

        Escribir "Categoría no válida";
```

```
            NuevoSueldo <- Sueldo;

        FinSegun

        //Mostrar resultado
        Escribir "El sueldo del trabajador con las horas extras es:";
        Escribir NuevoSueldo;

FinAlgoritmo
```

Ejercicio 5

La siguiente tabla expresa los costos diarios según el tipo de enfermedad, construye un algoritmo que calcule e imprima el costo total que representa un paciente, considerando que, si el paciente tiene de 14 a 22 años, se le agrega al costo total a pagar un 10% sobre el costo

Enfermedad	Costo paciente día
1	25
2	16
3	20
4	30

```
Algoritmo CostoEnfermedad

        //Introducción a la Programación
        //Programa 1: EstSelMul1.psc
        //Calcular costo total de paciente
        //Programa que calcula el costo total de un paciente
        //Autor: ® Fernando Jiménez Ávila © CCAP México

        //Declarar variables
        Definir Nombre Como CARACTER;
        Definir Costo, Dias, CostoTotal Como Real;
        Definir Edad, Enfermedad Como Entero;

        //Leer variables
        Escribir "Las enfermedades pueden ser 1, 2, 3 y 4";
        Escribir "Ingrese la enfermedad:";
        Leer Enfermedad;
        Escribir "Ingrese el nombre del paciente:";
        Leer Nombre;
        Escribir "Ingrese la edad del paciente:";
        Leer Edad;
        Escribir "Ingrese los días internados del paciente:";
        Leer Dias;

        //Evaluar Enfermedad según sea la condición
        Segun (Enfermedad) Hacer

                1:
                Costo <- Dias * 25;
                2:
                Costo <- Dias * 16;
                3:
                Costo <- Dias * 20;
                4:
                Costo <- Dias * 30;

        De Otro Modo:
```

```
            Escribir "Opción no válida";
            Costo <- Dias;

        FinSegun

        //Evaluar si Edad es igual o mayor a 14 y menor o igual a 22
        Si (Edad >= 14) Y (Edad <= 22) Entonces

            //Si se cumple con la condición, relizar cálculo
            CostoTotal <- Costo * 1.10;

            //Mostrar resultado
            Escribir "El costo total por los días en el hospital es:";
            Escribir CostoTotal;

        SiNo

            //Si no se cumple con la condición, mostrar mensaje
            Escribir "El costo total por los días en el hospital es:";
            Escribir Costo;

        FinSi

FinAlgoritmo
```

Problemas con estructuras de repetición

Estructura repetitiva Para

Ejercicio 1

Escribe un algoritmo que imprima el factorial de un número dado por el usuario.

```
Algoritmo Factorial

    //Introducción a la Programación
    //Programa 1: EstRepPar1.psc
    //Calcular el factorial de un número
    //Programa que calcula el factorial de un número
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Numero1, Resultado, Contador Como Entero;

    //Inicializar variable
    Resultado <- 1;

    //Leer variable
    Escribir "Ingrese el número del cuál desea obtener su factorial: ";
    Leer Numero1;

    //Iniciar ciclo desde 1 hasta Numero1 repeticiones
    Para Contador <- 1 Hasta Numero1 Con Paso 1 Hacer

        //Realizar cálculo, multiplicar Contador por Resultado
        Resultado <- Resultado * Contador;

    FinPara
```

```
    //Mostrar resultado
    Escribir "El factorial del número ingresado es: ";
    Escribir Resultado;

FinAlgoritmo
```

Ejercicio 2

Crea un algoritmo que imprima en pantalla los números del 1 al 100.

```
Algoritmo SerieCien

    //Introducción a la Programación
    //Programa 2: EstRepPar2.psc
    //Mostrar serie del 1 al 100
    //Programa que muestra una serie del 1 al 100
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Numero1 Como Entero;

    //Inicializar variable
    Numero1 <- 0;

    //Iniciar ciclo desder 1 hasta 100 repeticiones
    Para Numero1 <- 1 Hasta 100 Con Paso 1 Hacer

        //Mostrar resultado
        Escribir Numero1;

    FinPara

FinAlgoritmo
```

Ejercicio 3

Crea un algoritmo que calcule la tabla de multiplicar de cualquier número dado por el usuario.

```
Algoritmo TablaMultiplicar

    //Introducción a la Programación
    //Programa 3: EstRepPar3.psc
    //Calcular tabla de multiplicar
    //Programa que calcula la tabla de mutiplicar de un número dado
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Numero1, Contador, Resultado Como Entero;

    //Inicializar variables
    Numero1 <- 0;
    Contador <- 0;
    Resultado <- 0;

    //Leer Numero1
    Escribir "Ingrese la tabla de multiplicar a mostrar: ";
    Leer Numero1;

    //Iniciar ciclo desde 1 hasta 10 repeticiones
```

```
    Para Contador <- 1 Hasta 10 Con Paso 1 Hacer

        //Realizar cálculo
        Resultado <- Contador * Numero1;

        //Mostrar resultado
        Escribir Numero1 " X " Contador " = " Resultado;

    FinPara

FinAlgoritmo
```

Ejercicio 4

Escribe un algoritmo que calcule e imprima la suma de los primeros cien números.

```
Algoritmo SumaCien

    //Introducción a la Programación
    //Programa 4: EstRepPar4.psc
    //Calcular la suma de los cien primeros números
    //Programa que calcula la suma de los cien primeros números
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Suma, Contador Como Entero;

    //Inicializar variables
    Suma <- 0;
    Contador <- 0;

    //Iniciar ciclo desde 1 hasta 100 repeticiones
    Para Contador <- 1 Hasta 100 Con Paso 1 Hacer

        //Realizar la suma
        Suma <- Suma + Contador;

    FinPara

    //Mostrar resultado
    Escribir "La suma de los primeros cien números es: ";
    Escribir Suma;

FinAlgoritmo
```

Ejercicio 5

Escribe un algoritmo que imprima en pantalla las tablas de multiplicar de los números del 1 al 10.

```
Algoritmo TablaMultiplicar

    //Introducción a la Programación
    //Programa 5: EstRepPar5.psc
    //Calcular tablas de multiplicar del 1 al 10
    //Programa que calcula las tablas de multiplicar del 1 al 10
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Resultado, Contador, Tabla Como Entero;
```

```
    //Inicializar variables
    Resultado <- 0;
    Contador <- 0;
    Tabla <- 0;

    //Inicio del ciclo desde 1 hasta 10 repeticiones para la tabla
    Para Tabla <- 1 Hasta 10 Con Paso 1 Hacer

        //Inicio del ciclo desde 1 hasta 10 repeticiones para las multiplicaciones
        Para Contador <- 1 Hasta 10 Con Paso 1 Hacer

            //Realizar la multiplicación
            Resultado <- Contador * Tabla;

            //Mostrar resultado
            Escribir Tabla " X " Contador " = " Resultado;

        FinPara

    FinPara

FinAlgoritmo
```

Estructura repetitiva Mientras

Ejercicio 1

Escribir un algoritmo para imprimir los primeros 50 números impares

```
Algoritmo NumerosImpares

    //Introducción a la Programación
    //Programa 1: EstRepMie1.psc
    //Mostrar los primeros cincuenta números impares
    //Programa que muestra los primeros cincuenta números impares
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Numero1, Contador Como Entero;

    //Inicializar variables
    Numero1 <- 1;
    Contador <-0;

    //Inicio del ciclo hasta 50 repeticiones
    Mientras (Contador < 50) Hacer

        //Mostrar Numero
        Escribir Numero1;

        //Realizar cálculo de número impar
        Numero1 <- Numero1 + 2;
        //Realizar el acumulado o sumatoria del contador
        Contador <- Contador + 1;

    FinMientras

FinAlgoritmo
```

Ejercicio 2

Supongamos que debemos obtener la suma de los gastos que hicimos en nuestro último viaje, pero no sabemos exactamente cuántos fueron. Construye un algoritmo para conocer cuántos gastos se realizaron.

```
Algoritmo ViaticosViajes

    //Introducción a la Programación
    //Programa 2: EstRepMie2.psc
    //Calcular gastos de viaje
    //Programa que calcula el gasto total de un viaje
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Gasto, Total Como Real;

    //Inicializar variables
    Gasto <- 0;
    Total <- 0;

    //Leer variable
    Escribir "Ingrese el gasto: ";
    Leer Gasto;

    //Evaluar si Gasto es igual a 0 (cero)
    Si (Gasto = 0) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "No hubo gastos";

    SiNo

        //Si no se cumple con la condición, iniciar el ciclo hasta que Gasto sea igual a 0 (cero)
        Mientras (Gasto <> 0) Entonces

            //Realizar acumulado o sumatoria de Gasto
            Total <- Total + Gasto;

            //Mostrar el tota acumulado
            Escribir "El total es:";
            Escribir Total;
            //Leer Gasto
            Escribir "¿Hay más gastos? Ingresa la cantidad:";
            Leer Gasto;

            //Evaluar si Gasto es igual a 0 (cero)
            Si (Gasto = 0) Entonces

                //Si se cumple con la condición, mostrar mensaje
                Escribir "Ya no hay gastos";

            FinSi

        FinMientras

        //Mostrar resultado
        Escribir "El total es:";
        Escribir Total;
```

```
    FinSi

FinAlgoritmo
```

Ejercicio 3

Escriba un algoritmo dado un grupo de números naturales positivos, calcule e imprima el cubo de estos números.

```
Algoritmo CubosNumerosNaturales

    //Introducción a la Programación
    //Programa 3: EstRepMie3.psc
    //Obtener los cubos de los primeros nueve números naturales
    //Programa que calcula los cubos de los primeros nueve números naturales
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Numero1, Cubo Como Real;

    //Inicializar variables
    Numero1 <- 0;
    Cubo <- 0;

    //Leer Numero1
    Escribir "Ingrese el número natural: ";
    Leer Numero1;

    //Evaluar si Numero1 es menor a cero o es mayor a 9
    Si (Numero1 < 0) O (Numero1 > 9) Entonces

        //Si se cumple con la condición, mostrar mensaje
        Escribir "El número no es natural";

    FinSi

    //Iniciar el ciclo mientras que Numero1 sea igual o mayor que 0 y sea menor o igual que 9
    Mientras (Numero1 >= 0) Y (Numero1 <= 9) Entonces

        //Calcular el cubo del número
        Cubo <- Numero1 ↑ 3;

        //Mostrar el cubo
        Escribir "El cubo del número es:";
        Escribir Cubo;
        //Leer otro número natural
        Escribir "Ingresa otro número natural:";
        Leer Numero1;

        Evaluar si Numero1 es menor a cero o es mayor a 9
        Si (Numero1 < 0) O (Numero1 > 9) Entonces

            //Si se cumple la condición, mostrar mensaje
            Escribir "El número no es natural";

        FinSi

    FinMientras

FinAlgoritmo
```

Ejercicio 4

Calcule el aumento de sueldo para un grupo de empleados de una empresa teniendo en cuenta el siguiente criterio, si sueldo es inferior a $1,000.00, aumento del 15%; si sueldo es mayor o igual a $ 1,000.00, aumento del 12%. Imprima el sueldo nuevo del trabajador y el total de nómina de la empresa, considerando este nuevo aumento.

```
Algoritmo AumentoSueldo

    //Introducción a la Programación
    //Programa 4: EstRepMie4.psc
    //Cálculo de aumento y nómina
    //Programa que calcula el aumento de sueldo y la nómina total
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Trabajadores, Sueldo, SueldoNuevo, Nomina Como Real;

    //Inicializar variables
    Trabajadores <- 0;
    Sueldo <- 0;
    SueldoNuevo <- 0;
    Nomina <- 0;

    //Leer variables
    Escribir "¿Cuántos trabajadores tiene el grupo?";
    Leer Trabajadores;

    //Iniciar ciclo mientras Trabajadores sea mayor que 0 (cero)
    Mientras (Trabajadores > 0) Hacer

        //Leer sueldo
        Escribir "¿Cuál es el sueldo actual?";
        Leer Sueldo;

        //Evaluar si Sueldo es mayor que $ 1,000.00
        Si (Sueldo < 1000) Entonces

            //Si la condición se cumple, realizar cálculo
            SueldoNuevo <- Sueldo * 1.15;

        SiNo

            //Si la condición no se cumple, evaluar si Sueldo es mayor o igual que $ 1,000.00
            Si (Sueldo >= 1000) Entonces

                //Si la condición se cumple, realizar cálculo
                SueldoNuevo <- Sueldo * 1.12;

            FinSi

        FinSi

        //Calcular nómina
        Nomina <- SueldoNuevo * Trabajadores;

        //Mostrar resultados
        Escribir "El nuevo sueldo del grupo de trabajadores será de: ";
        Escribir SueldoNuevo;
        Escribir "El total de nómina es: ";
        Escribir Nomina;
```

```
        //Leer Trabajadores
        Escribir "¿Cuántos trabajadores tiene el grupo?";
        Leer Trabajadores;

    FinMientras

FinAlgoritmo
```

Ejercicio 5

Realiza un algoritmo que permita al usuario continuar dentro del sistema, donde pregunte: ¿Desea continuar?»; y como respuesta: S/N

```
Algoritmo OpcionContinuar

    //Introducción a la Programación
    //Programa 5: EstRepMie5.psc
    //Determinar si continuar o no continuar
    //Programa que determina si se continúa o no se continúa
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variable
    Definir Opcion Como Entero;

    //Inicializar variables
    Opcion <- 0;

    //Iniciar ciclo mientras Opcion sea diferente a 2
    Mientras (Opcion <> 2) Hacer

        //Mostrar mensaje
        Escribir "¿Deseas continuar?";
        //Leer Opcion
        Escribir "(1) Si (2)No";
        Leer Opcion;

    FinMientras

FinAlgoritmo
```

Ejercicio 6

Hacer un algoritmo para calcular la suma de los primeros cien números.

```
Algoritmo SumaNumeros

    //Introducción a la Programación
    //Programa 6: EstRepMie6.psc
    //Obtener la suma de los primeros cien números
    //Programa que calcula la suma de los primeros cien números
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
    Definir Numero1, Suma Como Entero;

    //Inicializar variables
    Numero1 <- 0;
    Suma <- 0;
```

```
    //Iniciar el ciclo de 100 veces
    Mientras (Numero1 <= 100) Entonces

        //Realizar cálculo, acumular la sumatoria de cada número
        Suma <- Suma + Numero1;
        //Acumular el contador
        Numero1 <- Numero1 + 1;

    FinMientras

    //Mostrar resultado
    Escribir "La suma de los primeros 100 números es: ";
    Escribir Suma;

FinAlgoritmo
```

Ejercicio 7

Crear un algoritmo que me muestre los números del 1 al 1,000

```
Algoritmo MilNumeros

    //Introducción a la Programación
    //Programa 7: EstRepMie7.psc
    //Mostrar números del 1 al 1,000
    //Programa que muestra los números del 1 al 1,000
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variable
    Definir Numero1 Como Entero;

    //Inicializar variable
    Numero1 <- 1;

    //Iniciar el ciclo de1,000 veces
    Mientras (Numero1 <= 1000) Hacer

        //Mostrar número
        Escribir Numero1;
        //Acumular contador
        Numero1 <- Numero1 + 1;

    FinMientras

FinAlgoritmo
```

Ejercicio 8

Escriba un algoritmo que obtenga la suma e imprima los términos de la siguiente serie: 2, 5, 7, 10, 12, 15, 17…1,800.

```
Algoritmo SerieNumeros

    //Introducción a la Programación
    //Programa 8: EstRepMie8.psc
    //Cálculo de serie n + 2, n + 3 y su suma
    //Programa que calcula las series de n + 2 y n + 3 y su suma
    //Autor: ® Fernando Jiménez Ávila © CCAP México

    //Declarar variables
```

```
Definir Serie, Contador Como Entero;

//Inicializar variables
Serie <- 0;
Contador <- 0;

//Mostrar mensaje
Escribir "A continuación, se imprime una serie de números de manera n + 2, n + 3";

//Inicio de ciclo para repetir 1,800 veces
Mientras (Serie < 1800) Hacer

//Realizar cálculos
//Obtener serie de n + 2 y acumular el contador
Serie <- Serie + 2;
//Acumular el contador para n + 2
Contador <- Contador + Serie;

//Mostrar la serie generada de n + 2
Escribir Serie;

//Obtener serie de n + 3 y acumular el contador
Serie <- Serie + 3;
//Acumular al contador para n + 3
Contador <- Contador + Serie;

//Mostrar la serie generada de n + 3
Escribir Serie;

FinMientras

//Mostrar resultado
Escribir "La suma de todos los términos de la serie es:";
Escribir Contador;

FinAlgoritmo
```

Estructuras combinadas

Ejercicio 1

Realizar un algoritmo para calcular la media de los números pares e impares, sólo se ingresarán 10 números.

```
Algoritmo MediaParesImpares

//Introducción a la Programación
//Programa 1: EstCom1.psc
//Cálculo de media de 10 números pares e impares
//Programa que calcula la media de 10 números pares e impares
//Autor: ® Fernando Jiménez Ávila © CCAP México

//Declarar variables
Definir Numero1, SumaPar, SumaImpar, Contador Como Entero;

//Inicializar variables
Numero1 <- 0;
SumaPar <- 0;
SumaImpar <- 0;
Contador <- 0;
```

```
    //Ciclo de 10 repeticiones
    Para Contador <- 1 Hasta 10 Con Paso 1 Hacer

        //Leer Numero1
        Escribir "Ingrese un número entero: ";
        Leer Numero1;

        //Evaluar si el residuo de Numero1 con 2 es 0 (cero)
        Si (Numero1 MOD 2 = 0) Entonces

            //Realizar cálculo, hacer la sumatoria
            SumaPar <- SumaPar + Numero1;

        SiNo

            //Realizar operación, realizar sumatoria
            SumaImpar <- SumaImpar + Numero1;

        FinSi

    FinPara

    //Realizar cálculos, obtener el promedio o la media
    SumaPar <- SumaPar / 10;
    SumaImpar <- SumaImpar / 10;

    //Mostrar resultados
    Escribir "La media de los números pares ingresados es: ";
    Escribir SumaPar;
    Escribir "La media de los números impares ingresados es: ";
    Escribir SumaImpar;

FinAlgoritmo
```