Introducción a la Programación y Computación 1 Sección E

Ing. MSc. Neftalí Calderón

Programación estructurada

El enfoque estructurado de programación constituye una forma simplificada de programación imperativa. La principal modificación del principio básico radica en que, en lugar de instrucciones de salto absolutas (instrucciones que provocan que el procesamiento no continúe con la siguiente instrucción), sino en otro lugar este paradigma de programación de software prevé el uso de bucles y estructuras de control.

Un ejemplo de ello es el uso de “do...while” para realizar una instrucción de forma automática siempre que se dé una determinada condición (al menos una vez).

```
instrucción1
instrucción2
instrucción2
instrucción4
.
.
.
.
.
```

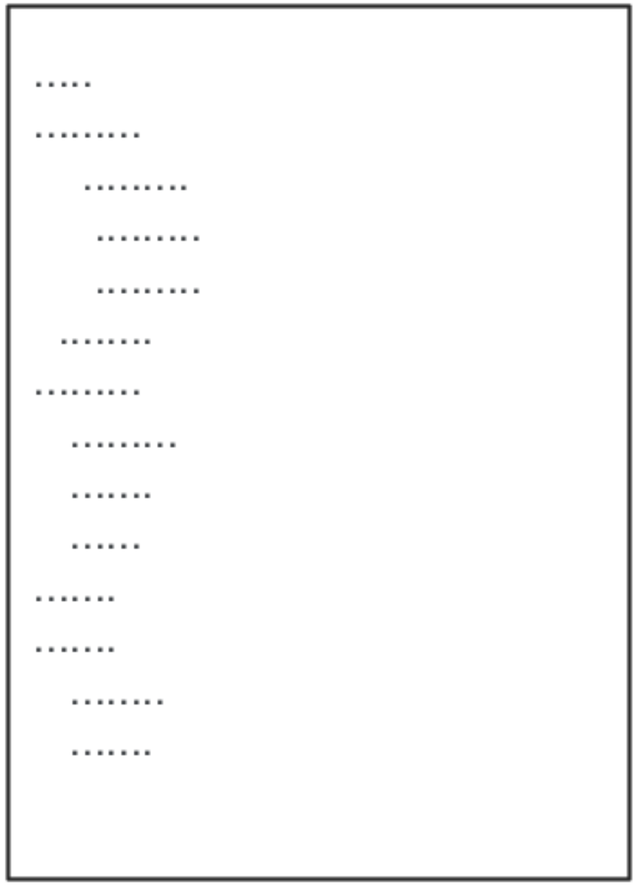
```
.
.
.
instrucciones

while (variable > 5) {
   instrucción1
   instrucción2
   instrucción2
   instrucción4
}

instrucciones
.
.
.
.
```

Programación procedimental

El paradigma de programación procedimental amplía el enfoque imperativo con la posibilidad de desglosar algoritmos en porciones manejables. Estos se denominan como procedimientos, dependiendo del lenguaje de programación, o también como subprogramas, rutinas o funciones.

El sentido y el propósito de esta distribución es hacer que el código de programa sea más claro y evitar las repeticiones innecesarias de código. Mediante la abstracción de los algoritmos, el paradigma de software procedimental representa un paso decisivo desde los lenguajes ensambladores simples hasta los lenguajes estándar complejos.

(a+b)*c/d+e-f



```
int precedencia(char op1) {
   if (op1 == '(') || (op1 == ')') {
       return(5)
    }
}
```

```
real suma(real n1, n2) {
    retum(n1+n2);
}
```

```
real multiplicacion(real n1, n2) {
   return(n1 * n2);
}
```

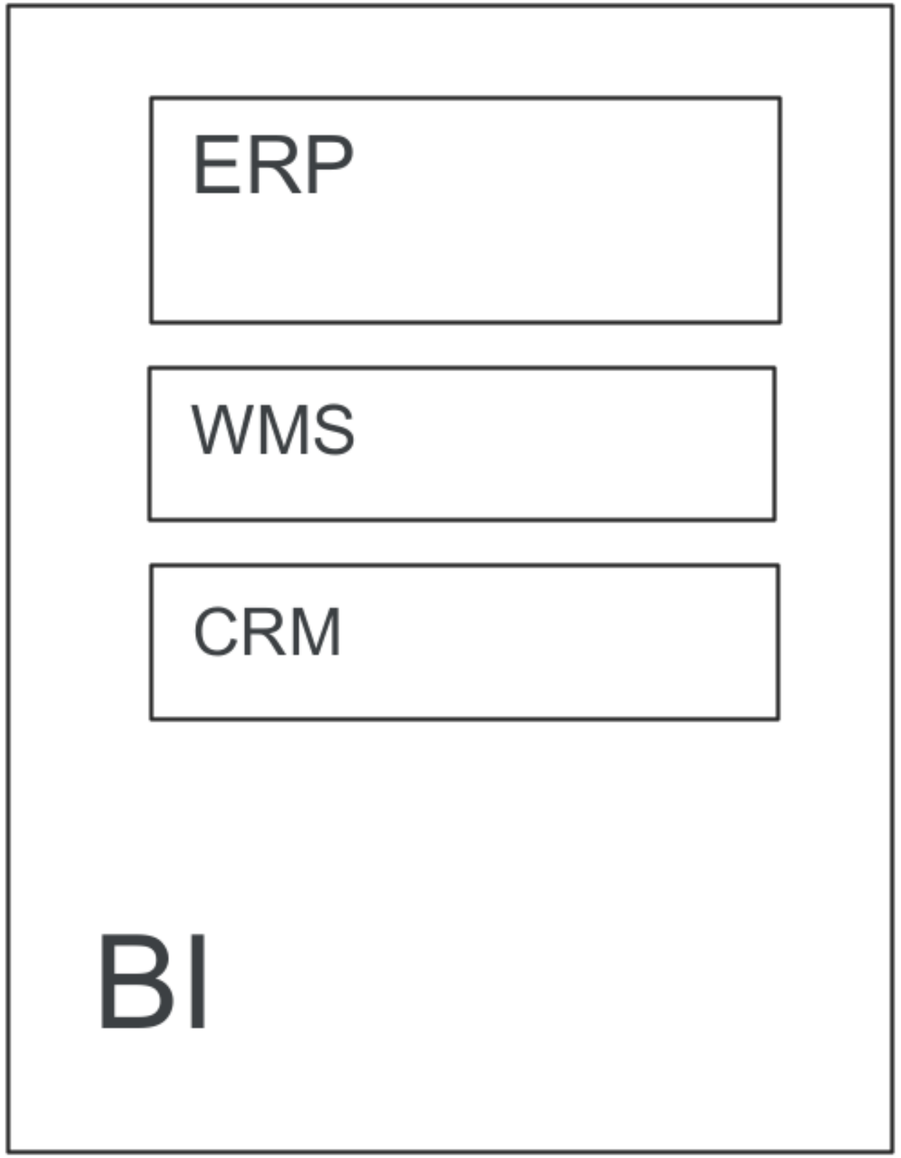
```
main() {
  leer(expresion);
  if ( (precedencia('*') > precedencia('+') ) {
     multiplicacion(b, c);
  }
}
```

Programación modular

La programación modular también se clasifica como un subtipo del paradigma de programación imperativo. En principio, es muy similar al enfoque procedimental, o más bien lo adapta a los requerimientos de proyectos de software mayores y más amplios.

En este sentido, el código fuente se divide específicamente en bloques parciales lógicos independientes los unos de los otros para proporcionar más transparencia y facilitar el proceso de debugging (resolución de errores). Los bloques parciales individuales, denominados módulos, se pueden probar por separado antes de vincularlos posteriormente a una aplicación conjunta.

ERP (facturación, CxC, CxP, Bancos, Inventario, RRHH, etc)


ERP
WMS
CRM
BI

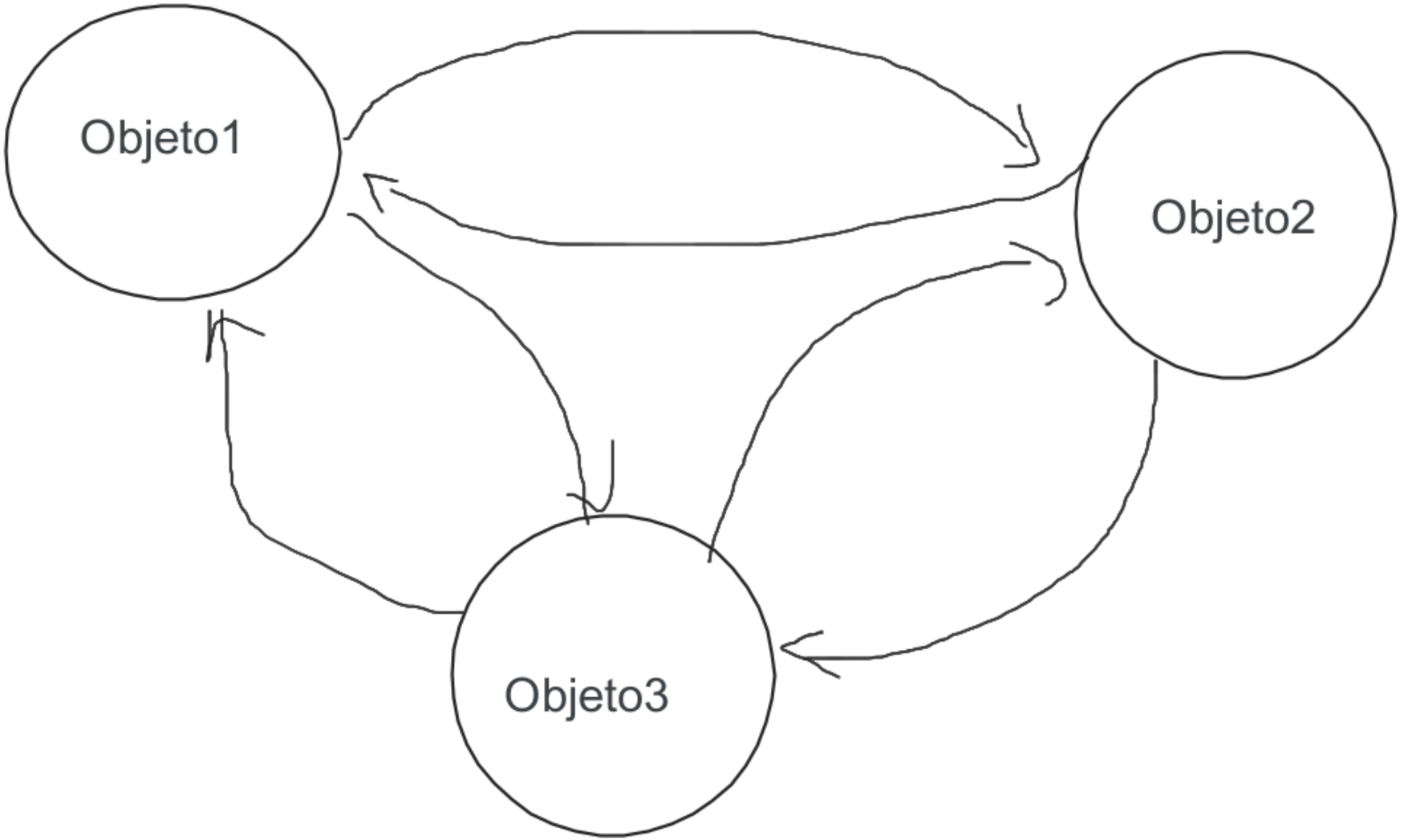
POO

La programación orientada a objetos viene de una filosofía o forma de pensar y analizar el problema en una forma de objetos para después llevarlo al código.

Se basa en 4 elementos y en 4 pilares:

Clases
Atributos
Métodos
Objetos

Encapsulamiento
Abstracción
Herencia
Polimorfismo


Objeto1
Objeto2
Objeto3

Programación funcional

Un programa de programación funcional consta de llamadas de función concatenadas en las que cada parte del programa se interpreta como una función.

En este sentido, las funciones dentro de la programación funcional pueden adoptar distintas “estructuras”. Por ejemplo, se pueden vincular entre sí como datos o se pueden utilizar en forma de parámetros. Asimismo, se pueden utilizar como resultados de función. En contraposición, el paradigma se ocupa de que no haya asignaciones independientes de valores.

cLisp (+(2,4,(*(3,5))))

Programación lógica (predicativa)

El paradigma de software lógico, denominado también como programación predicativa, se basa en la lógica matemática. En lugar de una sucesión de instrucciones, un software programado según este principio contiene un conjunto de principios que se pueden entender como una recopilación de hechos y suposiciones.

Todas las solicitudes al programa se procesan de forma que el intérprete recurre a estos principios y les aplica reglas definidas previamente para alcanzar el resultado deseado.

```
es_hijo(Juanito,Juan)
es_hijo(Juanito,Maria)
?es_hijo(x,Juan)
```

Tipos de datos primitivos

Entero (corto, largo)
Punto flotante (sencillo, doble)
Caracter
Booleano (falso / verdadero)
Cadena de caracteres

```
int miVariable = 123
char variable2 = 'g'
float varible3 = 41.5
string variable4 = 'Hola mundo'
```

Tipos de Datos	Memoria que ocupa	Rango de valores
boolean	1 byte	0 o 1 (True o False)
byte / unsigned char	1 byte	0 – 255
char	1 byte	-128 – 127
int	2 bytes	-32.768 – 32.767
word / unsigned int	2 bytes	0 – 65.535
long	2 bytes	-2.147.483.648 – 2.147.483.647
unsigned long	4 bytes	0 – 4.294.967.295
float / double	4 bytes	-3,4028235E+38 - 3,4028235E+38
string	1 byte + x	Array de caracteres
array	1 byte + x	Colección de variables

Constante

Una constante es un valor que no puede ser alterado/modificado durante la ejecución de un programa, únicamente puede ser leído.

var	int variable1
const	int IVA
	int IRS
	string EMPRESA
	string NIT

Pseudocódigo

Mezcla de lenguaje de programación y español (o ingles o cualquier otro idioma) que se emplea, para realizar el diseño de un programa.

El principal objetivo del pseudocódigo es el de representar la solución a un algoritmo de la forma más detallada posible, y a su vez lo más parecida posible al lenguaje que posteriormente se utilizará para la codificación del mismo.

Otra característica que tiene el pseudocódigo es su independencia al código en el que se va a escribir el programa, proporcionando un método que facilita la posterior programación y la resolución del algoritmo del programa.

Ventajas y desventajas

Las tareas más complejas o repetitivas pueden representarse de forma más sencilla ya que está escrito en un lenguaje sencillo y no estructurado que permite una transición sencilla al lenguaje de programación, más complejo y estructurado.

Tener un programa escrito en pseudocódigo facilita la tarea de programar en un lenguaje formal y mejora la calidad en la resolución de problemas, además de reducir el espacio necesario a la hora de desarrollar un problema.

No existen reglas claras para escribir pseudocódigo.

```
int altura;
int base;
int resultado = 0;

leer (base);
leer (altura);
resultado = base * altura / 2;
escribir('el área del triángulo es: ',resultado);
```

~~leer parametros;~~
~~calcular el área;~~
~~desplegar resultado;~~

```
int miFuncion(int dato1, dato2) {

}
```