



Universidad Tecnológica de Durango

Tecnologías de la Información

Fundamentos de programación

Actividades

“Evidencias de Actividades y Tareas”

Alumnos:

- Barraza Torres Jesús Daniel

1ºA BIS

Docente:

- Ing. Dagoberto Fiscal Gurrola, M.T.I.

Febrero 2025

Tabla de Ilustraciones

Ilustración 1. Decisiones compuestas, anidadas y múltiples	3
Ilustración 2. Representación de algoritmos con estructuras condicionales.....	4
Ilustración 3. Contador, acumulador y bandera.....	5
Ilustración 4. Mientras – Fin mientras (while)	6
Ilustración 5. Haga - Mientras que (do - while).....	7
Ilustración 6. Para - Fin para (for)	8
Ilustración 7. Representación de algoritmos.....	9

Actividad 1

Decisiones comp, anid, mult	9/2/25	9/2/25
<p>En la construcción de algoritmos, las decisiones compuestas, anidadas y múltiples son estructuras fundamentales que permiten la toma de decisiones y el control de flujo del programa.</p> <p>1. Decisiones compuestas Una decisión compuesta combina varias condiciones en una sola estructura de control, lo que permite evaluar más de una condición al mismo tiempo. -Características- Permite evaluar más de una condición a la vez. Se utiliza en sentencias como "if", "else-if" o "switch". Las condiciones pueden ser combinadas usando operadores lógicos que crean una decisión más compleja. -Usos- Determinar si cumplen varias condiciones al mismo tiempo, situaciones donde se requiere más de una condición para tomar una decisión, como validaciones combinadas.</p>		
<p>2. Decisiones anidadas Las decisiones anidadas ocurren cuando una sentencia de decisión está contenida dentro de otra. -Características- Se utiliza para evaluar condiciones dentro de otras condiciones. Las estructuras de control pueden estar dentro de otras. Son útiles para manejar situaciones con múltiples niveles de validación.</p>		
<p>3. Decisiones múltiples Las decisiones múltiples se refieren a una estructura que permiten elegir entre varias opciones, generalmente con la sentencia "switch" o utilizando múltiples "if-else". -Características- Permite seleccionar entre más de 2 o + opciones de manera clara y organizada. Generalmente se usan en estructura como "switch" (C o java) o múltiples sentencias "if-else". Evita la necesidad de anidar varias sentencias "if". -Usos- Seleccionar entre muchas opciones, o casos posibles (case en java). Implementación de sistemas de control que deben elegir entre múltiples alternativas.</p>		
<p>4. Decisiones simples Las decisiones simples son aquellas que involucran una pregunta continua de alternativas y no requieren una evaluación detallada de diferentes valores. -Características- Pocas alternativas. Fácil de implementar. (if, else if, switch, etc.) Lectura directa.</p>		
<p>-Usos</p> <ul style="list-style-type: none"> • Te permite poner límites en precisiones • Desarrollo en demás tipos de decisiones 		

Ilustración 1. Decisiones compuestas, anidadas y múltiples

En la imagen anterior se muestran las 4 categorías de las decisiones en la programación: Simples, compuestas, anidadas y múltiples; así como sus características y sus usos posibles dentro de la programación.

Actividad 2

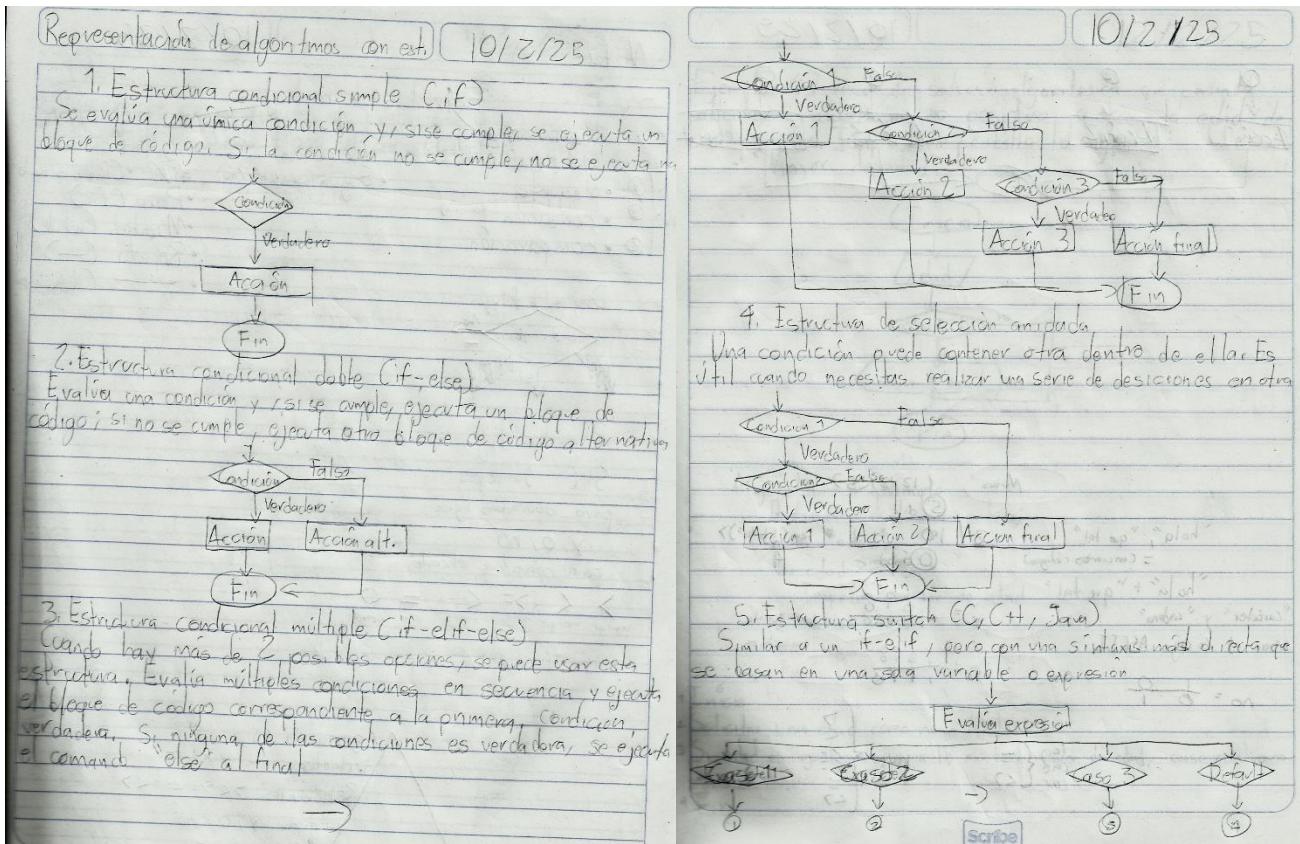


Ilustración 2. Representación de algoritmos con estructuras condicionales

En la imagen anterior se muestra una descripción simple de las 5 estructuras de control los cuales permiten la selección de decisiones, así como los diagramas de flujo respectivos de las siguientes estructuras condicionales: Simple, doble, múltiple, anidada y “switch”.

Actividad 3

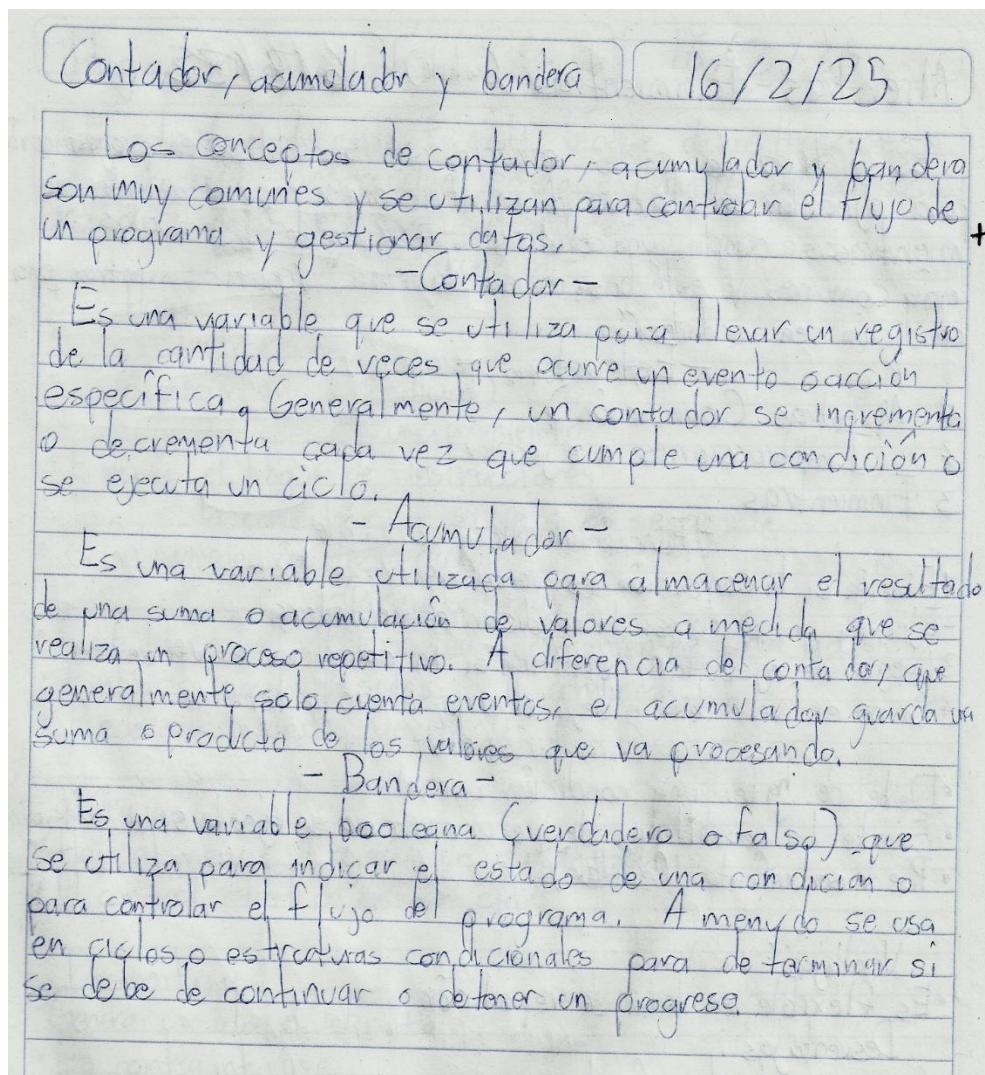


Ilustración 3. Contador, acumulador y bandera

En la imagen anterior se puede mostrar una descripción sencilla del contador, del acumulador y de la bandera, junto a su funcionamiento directo de cada uno de ellos y sus diferencias en caso del contador y del acumulador.

Actividad 4

Mientras – Fin mientras (while) [16/12/25]

Es un tipo de ciclo o bucle que se utiliza en programación para ejecutar un bloq que de instrucciones repetidamente mientras se cumple una condición. Es una de las bucles más comunes y está basada en la idea "repetir mientras una condición sea verdadera".

Estructura –

- 1 Mientras (Condición)
- 2 Instrucciones a ejecutar
- 3 Finmientras

- Funcionamiento –

- 1 Evalúa si la condición es verdadera
- 2 Ejecuta la acción si la condición es verdadera
- 3 Luego de ejecutar, verifica si la condición es verdadera
- 4 Si no es verdadera o deja de serlo, el ciclo termina.

- Características –

- Debe de haber una condición inicial
- Existe la posibilidad de que un ciclo no se ejecute si es falsa
- Puede ser infinito (evitar.)

Ventajas:

- Es flexible en tema de repetición

Desventajas:

- Genera un bucle infinito
- Si la iteración está mal, puede generar un error lógico

Ilustración 4. Mientras – Fin mientras (while)

En la imagen anterior se muestra las características de la función mientras – fin mientras, también conocido como “while”. A su vez, se puede ver la estructura, el funcionamiento, las características y sus ventajas y desventajas.

Actividad 5

Haga - mientras que (do-while)	16 / 2 / 25
<p>Es otra forma de ciclo. En este bloque de instrucciones, se ejecuta la instrucción dentro del bucle al menos una vez antes de evaluar su condición.</p>	
<p>-Estructura-</p>	
<ol style="list-style-type: none"> 1 Haga 2 Instrucciones 3 Mientras que (condición) 	
<p>-Funcionamiento</p>	
<ol style="list-style-type: none"> 1 Ejecuta el bloque de instrucciones 2 Evalúa la condición, y sigue si no se cumple 3 Si se cumple, repite la acción 	
<p>-Características-</p>	
<ul style="list-style-type: none"> • Tiene el control de ejecutar mínimo una acción • Evalúa la acción después de la condición • Puede ser infinito (avitar) 	
<p>Ventajas:</p>	
<ul style="list-style-type: none"> • Hay al menos 1 ejecución garantizada • El control repetitivo es más claro 	
<p>Desventajas:</p>	
<ul style="list-style-type: none"> • Genera un bloque infinito • Es contraintuitivo 	

Ilustración 5. Haga - Mientras que (do - while)

En la imagen anterior se muestra las características de la función haga – mientras que, también conocido como “do - while”. A su vez, se puede ver la estructura, el funcionamiento, las características y sus ventajas y desventajas.

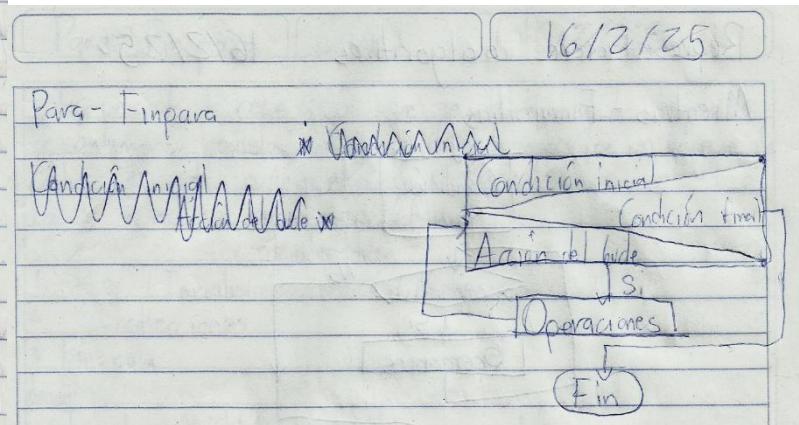
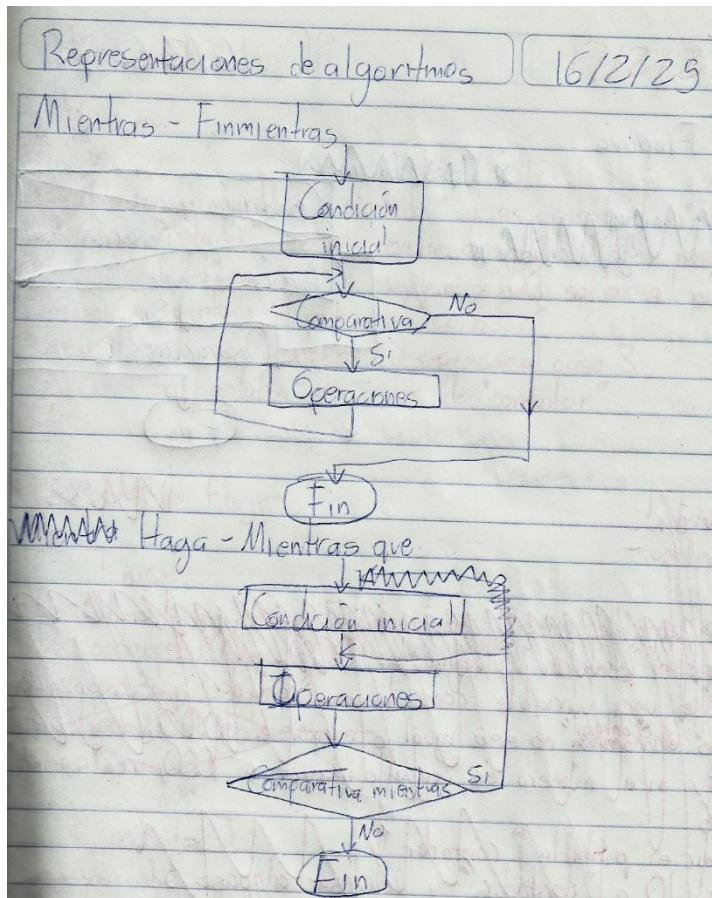
Actividad 6

Para - Fin para (for)	16/2/28
<p>Es un ciclo de repetición que se utiliza cuando se sabe de antemano cuántas veces se debe de ejecutar un bloque de código. Es útil cuando el nº de iteraciones es conocido o depende de una secuencia específica.</p> <p>-Estructura-</p> <ol style="list-style-type: none"> 1 Para (Inicialización; Condición; incremento) 2 Instrucciones 3 Fin para <p>Donde:</p> <ul style="list-style-type: none"> • Inicialización: Valor inicial de una variable • Condición: Expresión a evaluar (genera un booleano) • Incremento: Operación a la variable (generalmente un incremento) <p>-Funcionamiento-</p> <ol style="list-style-type: none"> 1 Se inicia la evaluación 2 Se evalúa si la condición es verdadera 3 Si la condición es verdadera, se ejecuta la instrucción 4 Al terminar la instrucción, se actualiza la variable 5 Se vuelve a ver, si <u>ese</u> vuelve falso, se termina el ciclo <p>-Características-</p> <ul style="list-style-type: none"> • Nº de iteraciones conocidas • Control preciso del índice • La condición es relativamente fácil de entender <p>Ventajas:</p> <ul style="list-style-type: none"> • Es simple • Es eficiente • Pasea un control automático <p>Desventajas:</p> <ul style="list-style-type: none"> • Poco flexible 	

Ilustración 6. Para - Fin para (for)

En la imagen anterior se muestra las características de la función para – fin para, también conocido como “for”. A su vez, se puede ver la estructura, el funcionamiento, las características y sus ventajas y desventajas.

Actividad 7



En esta actividad se puede observar la representación de los algoritmos Mientras – Fin mientras (while), Haga – Mientras que (do – while) y el Para – Fin para (for) gráficamente mediante las reglas utilizadas en un diagrama de flujo.

Retroalimentación

Estas actividades nos permitieron visualizar aquel tipo de algoritmo no secuencial el cual no fue descrito en la unidad anterior, siendo aquellos que se separan en múltiples ramas con acciones distintas cada uno o aquellos que regresan tras que una condición sea o no sea dada.

Este tipo de ejecuciones poseen mucha más utilidad que los algoritmos secuenciales presentados anteriormente podrían llegar a tener, permitiendo la fragmentación y desfragmentación de la ruta con propósitos de control y de repetición con la capacidad de resumir.

Esta actividad me ha permitido expandir mi razonamiento con la programación mediante la creación de rutas diferentes en caso de que ciertas condiciones se cumplan, mostrándome la capacidad de decisiones como un fundamento de la programación