MODULO

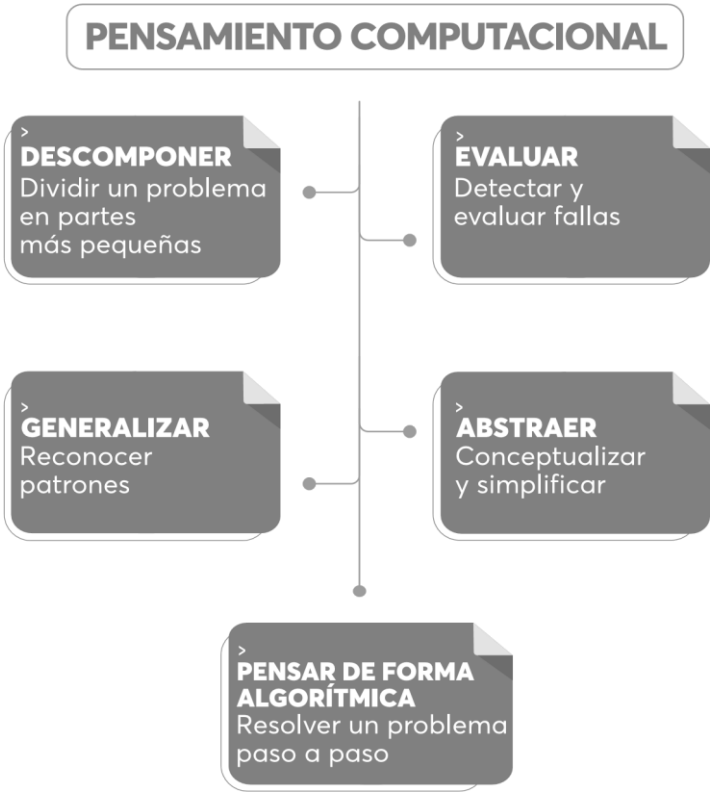
CICLO BASICO 3º AÑO

LENGUAJES TECNOLOGICOS III

TEMA: PENSAMIENTO COMPUTACIONAL

Pensamiento Computacional: Introducción

Texto Extraído de Cuadernillo "Introducción al Pensamiento Computacional" – UNIPE – EDUCAR



Los elementos clave del pensamiento computacional involucran el desarrollo de un razonamiento lógico. Este permite que los estudiantes puedan dar sentido a las cosas, lo que sucede por medio del análisis y la comprobación de los hechos a través de un pensamiento claro, detallado y preciso. De esta manera, los estudiantes toman sus propios conocimientos y modelos internos para hacer y verificar predicciones y así obtener conclusiones. El razonamiento lógico es la aplicación del pensamiento computacional para resolver problemas (CAS, 2015).

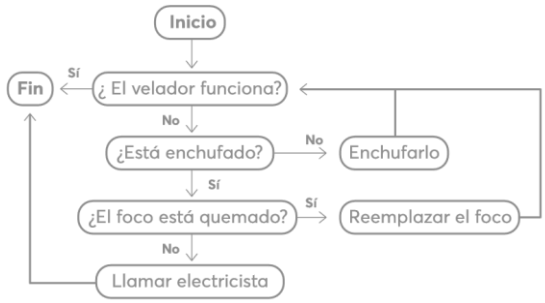
Por ejemplo, en actividades relacionadas con el diseño y la tecnología se aplica el razonamiento lógico —en particular, en el diseño de un objeto, la determinación de su forma y funcionalidad, la elección de los materiales a utilizar y los pasos de fabricación—. La descomposición se aplica al dividir un proyecto o proceso en diferentes partes según un criterio en particular. La generalización sucede cuando, dada una situación particular, el estudiante es capaz de establecer nuevas conexiones y pensar sobre otras aplicaciones o usos en otros contextos. Al diseñar, se están realizando tareas de predicción dado que se suponen comportamientos. El poder simular comportamientos implica evaluar situaciones futuras y así mejorar el diseño asegurando predicciones correctas. Cuando se escribe una solución a un problema, para que pueda ser implementada por una persona o una computadora, hay tareas relacionadas con la búsqueda y corrección de errores: en esa instancia, también existe un proceso de razonamiento.

El razonamiento lógico ayuda a explicar por qué sucede algo. Esto es muy importante en ciencias de la computación debido a que las computadoras son predecibles en sus resultados, solo realizan aquello para lo cual están programadas. En virtud de esta cualidad, se utiliza el razonamiento lógico para programarlas y así describir con exactitud las tareas a realizar. Entendido de esta manera, el razonamiento lógico equivale a explicar por qué algo es así.

CAPACIDAD DE PENSAR DE FORMA ALGORÍTMICA

Un algoritmo, en principio, es un objeto de comunicación compuesto por un conjunto finito de instrucciones que especifican una secuencia de operaciones concretas por realizar en un orden determinado para resolver un problema. El pensamiento algorítmico es una actividad cognitiva asociada a la resolución de problemas, a su especificación y a la comunicación de su solución.

Los siguientes son ejemplos de algoritmos que expresan soluciones a distintos problemas. Nótese que el algoritmo puede expresarse de distintas formas, en estos casos como un gráfico y como un texto con órdenes.



ALGORITMO PARA PREPARAR UNA SOPA INSTANTÁNEA EN EL HORNO DE MICROONDAS

1. inicio;
2. destapar el envase de la sopa;
3. agregar una taza pequeña de agua a la sopa;
4. introducir en el horno microondas;
5. programar el horno de microondas por 3 minutos;
7. fin.

En general, el pensamiento algorítmico se aplica cuando existen problemas semejantes que tienen que ser resueltos con periodicidad, entonces se analizan en conjunto y se desarrolla una solución general que se aplica cada vez que ocurre el problema.
En nuestra vida cotidiana, recurrimos de manera constante a algoritmos para solucionar problemas y así realizar cosas. Por ejemplo: para resolver una cuenta y obtener un valor, para cocinar una comida o para realizar una extracción de dinero en un cajero automático. En todos los casos mencionados, seguimos una y otra vez un conjunto ordenado de pasos que están almacenados en nuestro cerebro o en algún soporte externo (como en el caso de la receta de cocina que puede ser tomada de un libro o visualizada en YouTube).
Ejemplos de situaciones donde están presentes algoritmos:

- Cuando un cocinero escribe una receta para realizar un plato, está creando un algoritmo dado que otros pueden seguir los pasos y así reproducirla.
- Cuando un amigo anota las instrucciones para llegar a su casa, está especificando una secuencia de pasos (un algoritmo) para que otra persona lo pueda ubicar.
- Cuando un profesor proporciona un conjunto de instrucciones para llevar a cabo un experimento, está especificando un algoritmo, que es seguido por los estudiantes y así obtienen datos para su análisis y aprendizaje.

Podemos definir el pensamiento algorítmico como la capacidad de pensar en términos de secuencias y reglas que sirven para resolver problemas (CAS, 2015). Es un conocimiento básico que las personas desarrollan cuando aprenden a escribir sus propios programas de computadora, que no son otra cosa que algoritmos traducidos a instrucciones expresadas en un lenguaje que una computadora pueda comprender y ejecutar (por ejemplo, lenguajes informáticos como Scratch, Python, JavaScript, etc.).
Cuando se habla de software, se hace referencia directa a los datos digitales y a los programas de computadora. Lev Manovich, autor del libro *El software toma el mando*, realiza una descripción de su poder y su presencia en la sociedad actual cuando dice: «El *software* se ha vuelto nuestra interfaz con el mundo, con otras personas, con nuestra memoria e imaginación; un lenguaje universal mediante el cual habla el mundo, un motor universal mediante el cual funciona el mundo» (Manovich, 2013) y, por otro lado, nos advierte: «El *software* juega, hoy en día, un papel crucial en la confección de elementos materiales y de estructuras inmateriales que, aunados, constituyen la "cultura"».
En las ciencias de la computación, el trabajo de los científicos se concentra en encontrar los algoritmos más eficientes. Es decir, aquellos que resuelven un problema involucrando los menores recursos posibles (memoria, comunicaciones, tiempo de procesamiento, etc.) de la manera más efectiva al dar la respuesta correcta o la más cercana a ella.

Trabajo Práctico

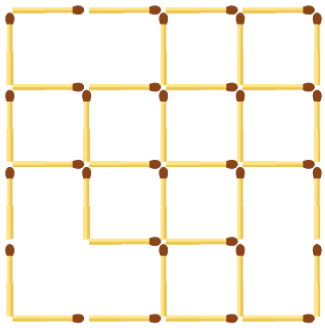
Actividad 1.

A partir de la lectura del Apunte teórico: ***pensamiento computacional teoría introductoria.pdf*** *responda las siguientes preguntas*
1.-Enumere los elementos claves del pensamiento computacional.
2.- Explique con sus palabras cada elemento del pensamiento computacional.
3.- ¿Cómo se aplica en el ámbito del diseño y la tecnología? (Contestar con tus palabras de acuerdo a lo que comprendes del texto leído)
4.- Luego de haber leído el texto explique con sus palabras que entiende por Razonamiento Lógico.
5.- ¿En qué consiste el pensamiento algorítmico? (Contestar con tus palabras de acuerdo a lo que comprendes del texto leído)
6.- ¿Cómo se relaciona el software y con los algoritmos? (Contestar con tus palabras de acuerdo a lo que comprendes del texto leído)

Actividad 2.

Resolver los siguientes desafíos:

Desafío 1: Contar cuadrados

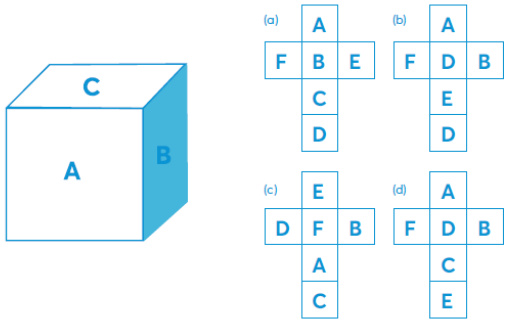


¿Cuántos cuadrados (de todos los tamaños) observas en el cuadrado?

Desafío 2: Próximo Vuelo

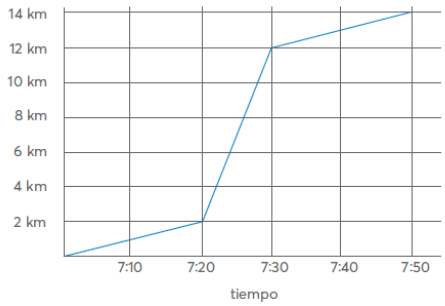
La empresa Aerolíneas B. utiliza un código único para identificar sus vuelos
El vuelo SLA 12 sale a las 06:15.
El vuelo SLA 15 sale a las 08:25.
El vuelo SLA 04 sale a las 12:10.
El vuelo SLA 08 sale a las 15:20.
¿cuál es el código para el vuelo que sale a las 18:00? ¿Cómo se construye un código de vuelo?

Desafío 3: Armar el cubo



¿Cuál de las figuras (a, b, c, o d) creará el siguiente cubo cuando se pliegue?

Desafío 4: Yendo a la escuela
Todos los días Juan sale de su casa y camina hacia la estación de tren, luego toma un tren hasta la estación cercana a su escuela y, finalmente, camina hacia esta. Su progreso se registra en el siguiente gráfico:



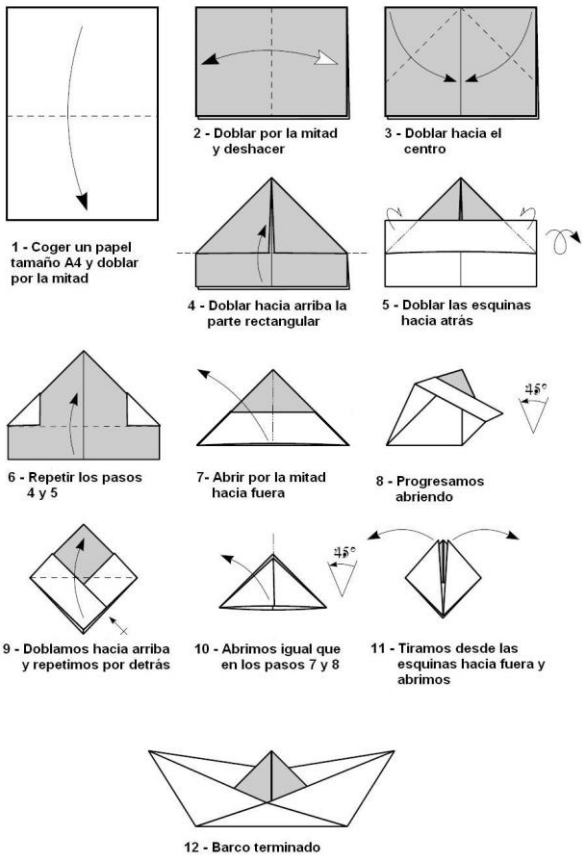
a) ¿A cuántos kilómetros de distancia se encuentra su escuela?
b) ¿Qué tan rápido (en km/h) camina Belén?
c) ¿Cuál es la velocidad promedio (en km/h) del tren?

TEMA: ALGORITMOS

Algoritmos

Un problema consiste en una situación que debe aclararse o resolverse y que puede tener un número determinado o indefinido de soluciones.
Cuando se habla de resolver problemas informáticos, es necesario aplicar procesos de razonamiento para asegurar que la solución obtenida sea la mejor. Para eso, se debe determinar cuál es la salida que se espera con respecto a los datos de entrada; o bien, buscar los datos de entrada que producen la salida o el resultado que se desea obtener.
Las computadoras tienen como una de sus funciones la resolución de problemas por medio de programas, por lo que se puede decir que éstos se construyen a través de un método para la solución de problemas.
Es por eso que el programador, a partir de una situación inicial, debe tener una idea sobre la solución sobre la solución que espera obtener para resolver determinado problema, y acerca del proceso que deberá plantearse para solucionarlo.
Para resolver un problema, ya sea en la vida diaria o de computación, se debe seguir una serie de pasos con el fin de llegar a un objetivo. A esa serie de pasos se le llama ***algoritmo***, el cual se define como un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.
Un ***algoritmo computacional*** se define como "un conjunto de operaciones y procedimiento que deben seguirse para resolver determinado problema en un entorno informático; y son el paso previo para la elaboración de un programa computacional.

Ejemplos:

A) Algoritmo para construir un barco de papel	*B) Algoritmo para un correcto lavado de manos (OMS)*
	• Mojarse las manos • Aplicar suficiente jabón para cubrir toda la mano • Frotar las palmas entre si • Frotar la palma de la mano derecha contra el dorso de la mano izquierda entrelazando los dedos , y viceversa • Frotar las palmas de las manos entre sí , con los dedos entrelazados • Frotar el dorso de los dedos de una mano contra la palma de la mano opuesta , manteniendo unidos los dedos • Rodeando el pulgar izquierdo con la palma de la mano derecha, frotarlo con un movimiento de rotación, y viceversa. • Frotar la punta de los dedos de la mano derecha contra la palma de la mano izquierda, haciendo un movimiento de rotación, y viceversa. • Enjuagar las manos. • Secarlas con una toalla de un solo uso.
	Algoritmo para calcular el área de un rectángulo (Area=base x altura)
	1. Inicio 2. Obtener medidas de base y altura 3. Multiplicar base por altura 4. Obtener resultado 5. Fin

CARACTERISTICAS DE LOS ALGORITMOS

Cuando se sigue una serie de reglas, pasos o instrucciones, éstas deben tener las siguientes características:

- **Objetivo**: Se debe conocer el final al que se quiere llegar con el algoritmo. Por lo tanto, debe resolver el problema para el que fue formulado.
- **Preciso**: Debe tener instrucciones claras para que sea un algoritmo preciso. Los resultados de los cálculos deben ser exactos, de manera rigurosa. No es válido un algoritmo que sólo se aproxime a la solución.

- **Finito**: Significa que cuenta con un determinado numero de paso, indicando un inicio y un fin, por lo que es conveniente numerar los pasos a seguir. No es valido aquel que no finalice.
- **Valido**: Debe carecer de errores a pesar de resolver.
- **Eficiente**: Se debe procurar que el algoritmo resuelva un problema procurando el menor número de pasos evitando redundancias que determinarán obtener soluciones en el menor tiempo.

Otro aspecto fundamental son los procedimientos, llamados operadores, con los que se lograr transformar la situación inicial hasta llegar a la solución final. Es decir, las operaciones permiten procesar la información.
Para el diseño de un algortimo en los programas computacionales se utilizan valores constantes, osea, datos que reciben un valor que no varía en todo el algoritmo (por ejemplo, π=3.14); y por otro lado, valores variables, que son datos asignados a un elemento que varía cuantas veces sea necesario durante el desarrollo del algoritmo. Por lo general, se representan con con 1 o varios caracteres alfanuméricos.

ELEMENTOS QUE CONFORMAN UN ALGORITMO

- **Entrada.** Los datos iniciales que posee el algoritmo antes de ejecutarse.
- **Proceso.** Acciones que lleva a cabo el algoritmo.
- **Salida**. Datos que obtiene finalmente el algoritmo.

¿TIPOS DE ALGORITMOS…?

Existen dos tipos y son llamados así por su naturaleza:

- Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.
- Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

CREACIÓN DE UN ALGORITMO

La computadora es una máquina que por sí sola no puede hacer nada, necesita ser programada, es decir, introducirle instrucciones u órdenes que le digan lo que tiene que hacer. Un programa es la solución a un problema inicial, así que todo comienza en el Problema. El proceso de programación es el siguiente: Dado un determinado problema el programador debe idear una solución y expresarla usando un algoritmo; luego de esto, debe codificarlo en un determinado lenguaje de programación y por último ejecutar el programa en el computador el cual refleja una solución al problema inicial.

TÉCNICAS DE DISEÑO DE ALGORITMOS DE PROGRAMACIÓN

TOP DOWN

También conocida como de arriba-abajo y consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al problema. Consiste en efectuar una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante entradas y salidas de información. Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.

La utilización de la técnica de diseño Top-Down tiene los siguientes objetivos básicos:

- Simplificación del problema y de los subprogramas de cada descomposición.
- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
- El programa final queda estructurado en forma de bloque o módulos lo que hace más sencilla su lectura y mantenimiento.

BOTTOM UP

El diseño ascendente se refiere a la identificación de aquellos procesos que necesitan computarizarse con forme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Entonces…

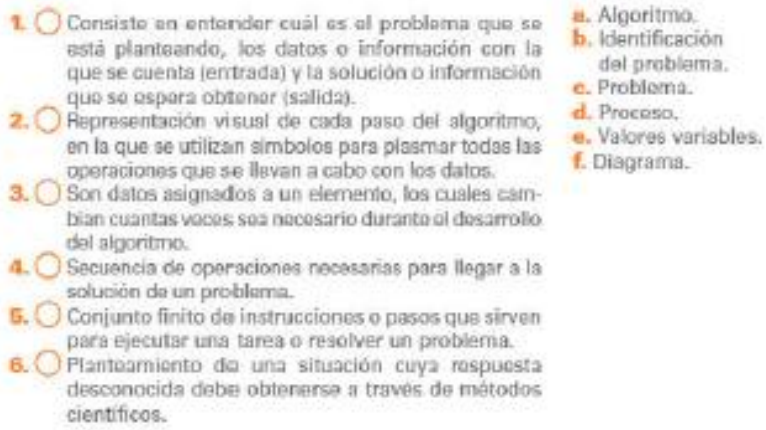
La diferencia entre estas dos técnicas de programación se fundamenta en el resultado que presentan frente a un problema dado. Imagine una empresa, la cual se compone de varios departamentos (contabilidad, mercadeo,…), en cada uno de ellos se fueron presentando problemas a los cuales se le dieron una solución basados en un enfoque ascendente (Bottom Up): creando programas que satisfacían sólo el problema que se presentaba. Cuando la empresa decidió integrar un sistema global para suplir todas las necesidades de todos los departamentos se dio cuenta que cada una de las soluciones presentadas no era compatible la una con la otra, no representaba una globalidad, característica principal de los sistemas. Como no hubo un previo análisis, diseño de una solución a nivel global en todos sus departamentos, centralización de información, que son características propias de un diseño Descendente (Top Down) y características fundamentales de los sistemas; la empresa no pudo satisfacer su necesidad a nivel global.

La creación de algoritmos es basado sobre la técnica descendente, la cual brinda el diseño ideal para la solución de un problema.

Trabajo Práctico

Actividad 1.

1) *Relaciona las definiciones de la columna izquierda (identificada con números) y los conceptos de la columna derecha (identificada con letras).*



2) *¿Qué es un problema? Menciona uno y enumera los pasos para resolverlo.*

Actividad 2.

A partir de la lectura del apunte y/o de recabar información del video sugerido, responder las siguientes preguntas:

1.- ¿Cómo se deben aplicar los procesos de razonamiento para la resolución de problemas?

2. -¿Qué es un algoritmo?

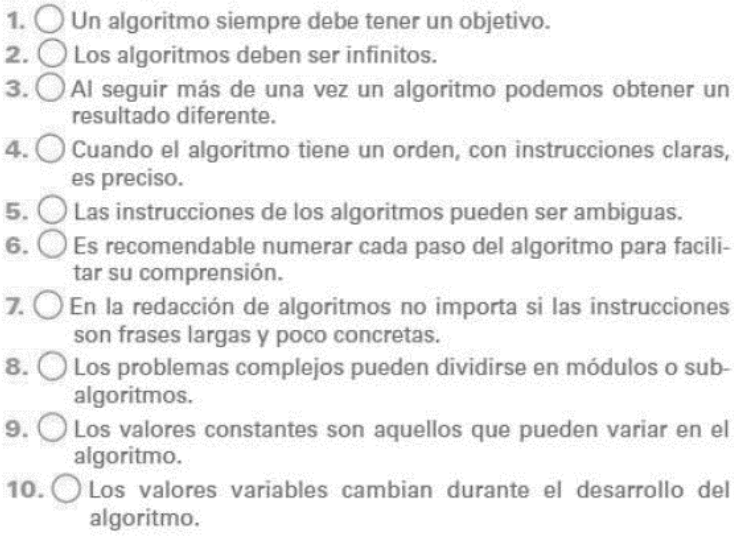
3.- ¿A qué se llama algoritmo computacional?

4.- ¿Cuáles son las etapas que componen la metodología para la resolución de problemas?

5.- *¿Cuáles son las diferencias de entre las distintas técnicas de diseño de algoritmos de programación?*

Actividad 3.

1) Leer las siguientes oraciones y determinar si cada una de ellas es Verdadera (V) o Falsa (F). En el caso de ser Falsas, justificarla.



Actividad 4.

Definir un algoritmo para realizar cada una las siguientes situaciones problemáticas

a) Ingresar al classroom de esta materia

b) Descargar una imagen desde internet, insertarla en el programa paint y guardarla en una carpeta con formato .png partiendo desde que se enciende la computadora.

c) Calcular el promedio de tres calificaciones

d) Calcular el área de un triángulo

METODOLOGÍA DE RESOLUCIÓN DE PROBLEMAS

- DEFINICIÓN DEL PROBLEMA: Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.
- ANÁLISIS DEL PROBLEMA: Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:
  - Los datos de entrada.
  - Cual es la información que se desea producir (salida)
  - Los métodos y fórmulas que se necesitan para procesar los datos.

LENGUAJES ALGORÍTMICOS

Un Lenguaje algorítmico es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso y facilita el diseño del algoritmo.

Tipos de Lenguajes Algorítmicos

- **Diagramas de Flujo**

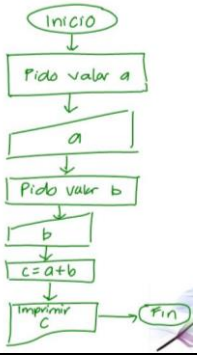
Los diagramas de flujo, como lo dice su nombre, representan el flujo de operaciones definidas en un algoritmo. Se utiliza una simbología estandarizada para facilitar la comprensión de un algoritmo. En los D.F los símbolos se conectan por medio de líneas de flujo lo cual indica que el orden de los símbolos es fundamental para una correcta interpretación del algoritmo.
El diseño del D.F es la interpretación que hacen los programadores antes de realizar el programa en un lenguaje de programación.

Símbolo	Significado
	Inicio/Fin. Determina el inicio y fin del algoritmo.
	Entrada por teclado. Representa el ingreso de los datos al programa.
	Proceso. Representa las operaciones que se efectúan para obtener el resultado.
	Decisión. Representa las operaciones de tipo lógico que contenga el algoritmo.
	Salida por impresora. Se utiliza cuando se desea obtener el resultado en papel.
	Salida por pantalla. Se utiliza cuando solamente se va a mostrar el resultado en pantalla.
	Conector. Se utiliza para conectar bloques del diagrama cuando el diagrama es grande y es necesario dividirlo.
	Líneas de flujo. Indican la secuencia del flujo de operación del diagrama.

- **Pseudocódigo**

El pseudocódigo tiene relación con los lenguajes de programación en que al ser una serie de palabras que están entre la sintaxis de un lenguaje de programación y la lengua coloquial. Se encuentra entre el diseño del algoritmo, el cual contiene palabras coloquiales y el programa informático, el cual está conformado por una serie de instrucciones estructuradas en una sintaxis determinada. Es la etapa intermedia entre el algoritmo y el programa final y se puede utilizar independientemente del lenguaje de programación que use el programador.

EJEMPLO: METODOLOGÍA Y LENGUAJES ALGORÍTMICOS
El requisito para tramitar la Credencial o Licencia de conducir es que las personas tengan 18 años cumplidos. Plantea la solución a través de un pseudocódigo para que se pueda autorizar la tramitación de la misma.

ALGORITMO: Sumar dos números		
Análisis	*Pseudocódigo*	*Diagrama de Flujo*
Datos de entrada: Los números a sumar: a y b. Datos de salida: Resultado de la suma: c Fórmula: c=a+b	1. Pedir valor a 2. Pedir valor b 3. c= a+b 4. Imprimir c	

Trabajo Practico

Actividad 1.

Relacionar correctamente ambas columnas anotando en el círculo el inciso que corresponda

1. ◯ Inicio/Fin.
2. ◯ Entrada por teclado.
3. ◯ Proceso.
4. ◯ Decisión.
5. ◯ Salida por impresora.
6. ◯ Salida por pantalla.
7. ◯ Conector.
8. ◯ Líneas de flujo.

a. 
b. 
c. 
d. 
e. 
f. 
g. 
h. 

Actividad 2.

1*.- Analizar la siguiente situación problemática y elaborar el algoritmo diseñando el D.F.D que le corresponde según los elementos simbólicos sugeridos el cual diseña un algoritmo que hace referencia al requisito para tramitar la Credencial o Licencia de conducir es que las personas tengan 18 años cumplidos. Plantear la solución en diseño de Flujo de Datos para que se pueda autorizar la tramitación de la misma teniendo en cuenta la siguiente simbología:*

b) Modificar el ejercicio anterior a fin de incluir en la autorización la aprobación del examen de conducir.

2.- *Plantear la solución para convertir determinado valor numérico de grados Fahrenheit a grados Centígrados, contemplando la posibilidad de que si la temperatura es mayor a 30ºC se muestre en pantalla un mensaje que diga "Día caluroso" de lo contrario que muestre "Día agradable". La fórmula de convertir ºF a ºC es ºC=5/9(ºF-32)*

Actividad 3.

Aplicando lenguajes algorítmicos, resuelva las siguientes situaciones problemáticas

1.- Una empresa de computadoras ofrece en venta computadoras a precio promocional de 25% de descuento del precio de lista si se compran 3 o más computadoras.
En el caso de que la venta no cumpla con las condiciones de la promoción el precio de venta de cada computadora será el precio de lista. Desarrollar un pseudocódigo que desarrolle un algoritmo que permita:

- *Ingresar precio de lista de venta de computadoras*
- *Ingresar la cantidad de computadoras*
- *Si la cantidad de computadoras es mayor o igual a tres, calcular descuento y el precio a pagar por el cliente*
- *Mostrar al cliente el monto a pagar*

Estructuras de Control

Para lograr que un algoritmo llegue a la solución se pueden usar distintas estructuras. Las secuencias, o también llamadas estructuras de control tienen como objetivo ofrecer diferentes opciones de solución dependiendo de determinadas condiciones, las cuales se conocen como sentencias alternativas o selectivas o de decisión, ya que se pueden elegir, de entre varias la ejecución de ciertas sentencias.
Otro de los objetivos de las estructuras de control es poder ejecutar un proceso varias veces hasta que se cumpla determinada condición; a estos procesos, en el ambiente de programación, se los conoce como bucles.
Cuando un algoritmo no tiene alternativas de selección o procesos repetitivos se le conoce como flujo secuencial, ya que se ejecutan línea tras línea las instrucciones del proceso hasta llegar a su final.
Las estructuras de control son:
Estructura Secuencial: Sigue el orden de las instrucciones planteadas en el algoritmo, por lo que existe un solo camino para obtener el resultado. Por ejemplo, calcular el importe de un determinado número de artículos vendidos del mismo tipo.

1. ***Inicio***
2. ***Leer cantidad de artículos (CA)***
3. ***Leer Precio (P)***
4. ***IMPORTE = CA * P***
5. ***Mostrar IMPORTE***
6. ***Fin***

Estructura Selectiva o de decisión: El algoritmo cuenta con dos alternativas, de las cuales se selecciona una, dependiendo del resultado que se obtenga. Por ejemplo, calcular el importe de un determinado número números de artículos. Si el importe es mayor de $6000, aplicar un descuento del $500.

1. ***Inicio***
2. ***Leer cantidad de artículos (CA)***
3. ***Leer Precio (P)***
4. ***IMPORTE = CA * P***
5. ***Si IMPORTE >6000 entonces***
   TOTAL=Importe - 500
   Sino
   TOTAL= IMPORTE
6. ***Mostrar TOTAL***
7. ***Fin***

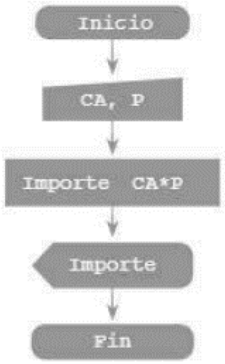
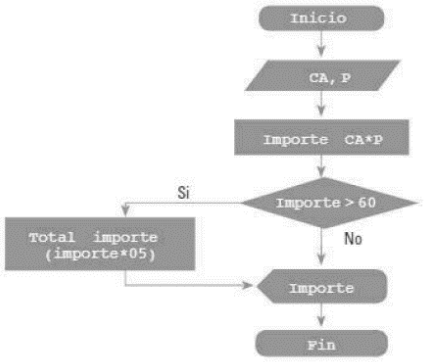
Estructura Repetitiva, iterativa, ciclo o bucle: Cuando el algoritmo requiere que un conjunto de operaciones o instrucciones se realicen un número finito de veces, entonces las instrucciones se efectúan un número determinado de veces o mientras una condición sea cierta o verdadera.

Ejemplo 1 – Número finito de veces: Calcular el importe de un determinado número de artículos vendidos del mismo tipo. Realizar esta operación 5 veces.	***Ejemplo 2: Según condición verdadera*** Calcular el importe de un determinado número de artículos vendidos del mismo tipo. Realizar esta operación tantas veces como el vendedor lo requiera.
1. ***Inicio*** 2. ***Para 1 hasta 5*** ***Leer cantidad de artículos (CA)*** ***Leer Precio (P)*** ***IMPORTE = CA * P*** ***Mostrar IMPORTE*** 3. ***Fin Para*** 4. ***Fin***	1. ***Inicio*** 2. ***Mostrar "Realiza Venta?"*** 3. ***Leer RESPUESTA*** 4. ***Mientras RESPUESTA = "SI"*** ***Leer cantidad de artículos (CA)*** ***Leer Precio (P)*** ***IMPORTE = CA * P*** ***Mostrar IMPORTE*** ***Mostrar "Realiza otra Venta?"*** ***Leer RESPUESTA*** 5. ***Fin Mientas*** 6. ***Fin***

Trabajo Práctico

Actividad 1.

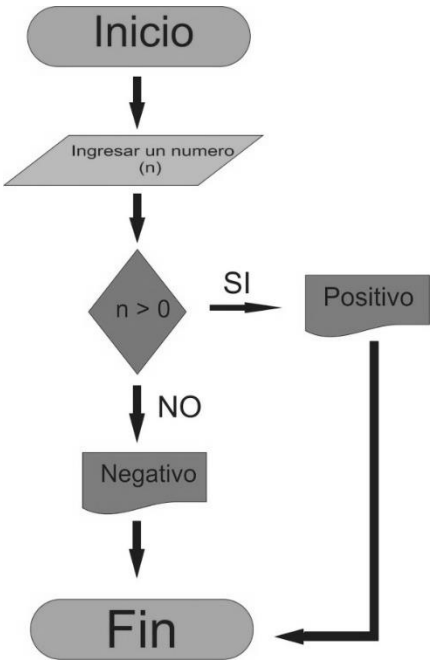
Observar los siguientes Enunciados, Diagramas de Flujo, algoritmos o pseudocódigo e identificar que secuencia de control se implementaría en cada uno.

Ejercicio A	*Ejercicio B*	*Ejercicio C*	*Ejercicio D*
	***1. Inicio*** ***2. Leer NUMERO1*** ***3. Leer NUMERO2*** ***4. Si NUMERO1>NUMERO2 entonces MAYOR=NUMERO1 Sino MAYOR=NUMERO2*** ***5. Mostrar MAYOR*** ***6. Fin***	***Calcular y mostrar el promedio anual de 15 alumnos ingresando para cada uno las tres notas trimestrales***	

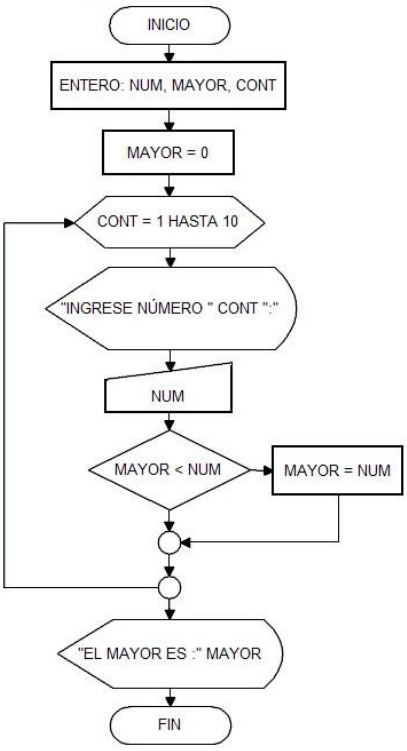
Actividad 2.

Dados los siguientes algoritmos en Lenguaje de Diagrama de Flujo:

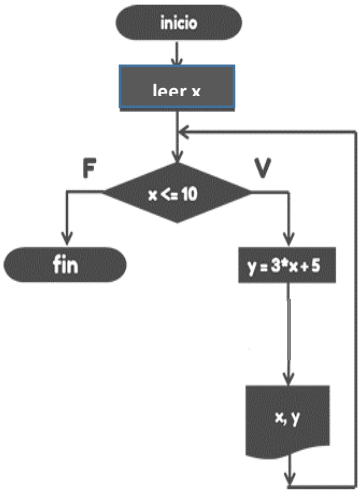
A



B



C



1) ¿Cuál es el objetivo de cada algoritmo?
2) ¿Qué estructuras de control implementa cada uno?
3) ¿Cuál sería el enunciado del problema que se resolvería con cada uno de los D.F.D?

TEMA 3: SENTENCIAS SIMPLES Y ESTRUCTURAS DE DECISIÓN C++

Un programa en C++ (o en C) consta de una o más **funciones**, las cuales están compuestas de diversas **sentencias** o **instrucciones**. Una sentencia indica una acción a realizar por parte del programa. Una función no es más que (por ahora) un nombre con el que englobamos a las sentencias que posee a fin de poder invocarlas mediante dicho nombre.

ESTRUCTURA DE UN PROGRAMA EN C++

Comentarios →	// Programa /*Ejemplo*/
	#include<iostream>
Directivas del preprocesador →	*#include<……>*
	using namespace std;
main( ) // Función principal *main* →	*main()*
{ //Inicio de la Función →	*{*
	*char **nombre[10];***
Declaración variables de la función principal →	*int **edad;***
	*float **altura;***
	cout<<"Cual es tu nombre?"<<endl;
	cin>>nombre;
	cout<<"Cual es tu edad?"<<endl;
Instrucciones de la función principal →	*cin>>edad;*
	cout<<"Cual es tu altura?"<<endl;
	cin>>altura;
	cout<<"Hola "<<nombre<<endl;
return sin ~~errores~~; →	*return 0;*
} //Fin de la Función Principal	*}*

ELEMENTOS DE UN PROGRAMA EN C++.

- **Sentencias**: son los elementos básicos en los que se divide el código de C++. En realidad, un programa es un conjunto de **sentencias** que se ejecutan para realizar una cierta tarea. Además, toda sentencia termina con punto y coma indicando el fin de una y el comienzo de la siguiente. Además, deben escribirse en minúscula excepto lo que se imprime o muestra en pantalla y el nombre de las variables.
- **Comentarios:** Se trata de texto que es ignorado por el compilador al traducir el código. Esas líneas se utilizan para documentar el programa. Se pueden delimitar /* y */ o comentarios por línea con la simbología //
- **Líneas de preprocesador:** comienzan con el símbolo # y que no finalizan con punto y coma. Son instrucciones que serán traducidas a código fuente antes de iniciar el proceso de compilado. La más utilizada es #include.
- **Función Principal main():** es la función cuyas instrucciones se ejecutan cuando el programa inicia su proceso.
- **Declaración de variables:** En C++ hay que declarar las variables antes de poder utilizarlas, ya que de esta forma se reserva en memoria el espacio para almacenar el contenido de la variable según su tipo de dato. **(*Ver el apartado VARIABLES)***
- **Identificadores:** nombre de variables y funciones.
- **Palabras reservadas propias del Lenguaje C++**
- **La sentencia using namespace std:** invoca a la librería estándar.

VARIABLES

Para definir o declarar una variable hay que utilizar una expresión del tipo:

Tipo_de_dato nombre_variable; //donde tipo es un tipo válido y nombre_variable es el //identificador de la variable

TIPOS DE DATOS BÁSICOS C++

Tipo de dato	Notación en C/C++	Tipo de dato	Notación en C/C++
entero	***int***	real	***float*** y ***double***
carácter	***char***	Lógico o booleano	***bool***

OPERADORES

Son elementos que relacionan de forma diferente, los valores de una o más variables y/o constantes. Es decir, los operadores nos permiten manipular valores.

Operadores Aritméticos	***Operadores Relacionales***	***Operadores Lógicos***
Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes). Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.	Se utilizan para establecer una relación entre dos valores. Luego compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso). Los operadores relacionales comparan valores del mismo tipo (numéricos o cadenas). Estos tienen el mismo nivel de prioridad en su evaluación. Los operadores relaciónales tiene menor prioridad que los aritméticos.	Estos operadores se utilizan para establecer relaciones entre valores lógicos. Estos valores pueden ser resultado de una expresión relacional.
+ Suma - Resta * Multiplicación / División % Modulo (resto de la división entera)	> Mayor que < Menor que > = Mayor o igual que < = Menor o igual que ! = Diferente o distinto = = Igual	&& Y || O

Trabajo Práctico

Actividad

a) ¿Qué es una variable? ¿Por qué es necesario declararla en el programa?
b) ¿A qué se llama tipo de Dato y por qué es importante su utilización en programación?
c) ¿Por qué en todo programa de C++ se debe incluir la función main()?

SENTENCIAS SIMPLES

SENTENCIAS DE ASIGNACIÓN En C++ el ***operador de asignación*** de una expresión a una variable es el carácter ***=.***	***Sintaxis y ejemplo:*** int A; // Se declara la variable A del tipo entero A=6; // A la variable A se le asigna el valor 6.
SENTENCIAS DE LECTURA O INGRESO DE DATOS Se utiliza la instrucción **cin** cuya función en guardar en la variable el valor ingresado por teclado	***Sintaxis***: *cin>>nombre_variable;*
SENTENCIAS DE ESCRITURA O VISUALIZACIÓN POR PANTALLA Se utiliza la instrucción cout cuya función es mostrar por pantalla.	***Sintaxis y Ejemplos:*** *cout<<nombre_variable* //muestra el contenido de la variable *cout<<"FRASE";* *//muestra lo que esta entre comillas* *cout<<"El contenido de la variable es "<<nombre_variable<<endl;* //muestra la frase entre comillas y el contenido de la variable

EJEMPLO DE ANÁLISIS Y CODIFICACION EN C++ CON SENTENCIAS SIMPLES

En este ejemplo, se pretende mostrar como codificar un programa en C++ que permita calcular el área de un rectángulo, cuya fórmula es: Area=Base x Altura, y cuyo resultado deberá mostrarse por pantalla. Para ello, debemos realizar

1) Análisis del Problema:

- *¿Qué datos se necesitan para calcular el Área del triángulo?* Los valores de la base y la altura del rectángulo
- *¿Qué variables se usarán, cómo se identificarán, y de qué tipo serán?*
  Las variables recibirán en nombre de:
  - **altura**, donde se guardará la altura del rectángulo, cuyo tipo de dato será ***float***, porque puede tomar valores numéricos con decimales.
  - **base**, donde guardará la base del rectángulo, cuyo tipo de dato será ***float***, porque puede tomar valores numéricos con decimales.
  - **area**, donde guardará el resultado de la fórmula, cuyo tipo de dato será ***float***, porque puede tomar valores numéricos con decimales.
- *¿Cómo se ingresaran los datos necesarios para el cálculo y qué valores se mostrarán?* Los valores de la base y de la altura se ingresaran por teclado. El valor obtenido al calcular el area se mostrará por pantalla.

2) Codificación en C++:

EXPLICACIÓN	*CODIGO C++*
Comentarios	// Programa Ejemplo /*Calculo del area del rectangulo*/
Directivas del preprocesador	*#include<iostream>* *using namespace std;*
Función principal *main()*	*main()*
Inicio de la Función Principal	*{*
Declaraciones locales de la función principal	*float **BASE, ALTURA, AREA;***
Escribe (o muestra) en pantalla lo que esta entre comillas	*cout<<"*********"<<endl;*
Escribe (o muestra) en pantalla lo que esta entre comillas	*cout<<"CALCULO AREA DE UN RECTANGULO"<<endl;*
Escribe (o muestra) en pantalla lo que esta entre comillas	*cout<<"*********"<<endl;*
Muestra (o muestra) en pantalla lo que esta entre comillas	*cout<<"Ingrese la base: "<<endl;*
Lee el dato ingresado por teclado guardándolo en variable BASE	*cin>>BASE;*
Muestra (o muestra) en pantalla lo que esta entre comillas	*cout<<"Ingrese la altura: "<<endl;*
Lee el dato ingresado por teclado guardándolo en variable ALTURA	*cin>>ALTURA;*
Aplica la fórmula calculando el área	*AREA=(BASE*ALTURA;*
Muestra (o escribe) en pantalla lo que esta entre comillas junto con el contenido de la variable AREA,	*cout<<"El área del rectangulo es "<<AREA<<endl;*
Escribe (o muestra) en pantalla lo que esta entre comillas	*cout<<"*********"<<endl;*
La funcion main() retorna 0;	*return 0;*
Fin de la Función Principal	*}*

Estructura if - else

Los pasos o instrucciones que se encuentran entre el inicio y fin de la estructura, tienen dos secciones:

1. Un conjunto de **instrucciones** que se ejecutan cuando la evaluación de la **condición** sea **verdadera**
2. Un conjunto de **instrucciones** que se ejecutan cuando la evaluación de la **condición** sea **falsa**.

if Sentencia Condicional Simple	**If-else** Sentencia Condicional Compuesta	**If-else-if** Anidación
Se trata de una sentencia que, tras evaluar una expresión lógica, ejecuta una serie de sentencias en caso de que la expresión lógica sea verdadera.	Es similar a la sentencia condicional simple, sólo que se añade un apartado else que contiene instrucciones que se ejecutarán si la expresión evaluada por el if es falsa.	Dentro de una sentencia if se puede colocar otra sentencia if. A esto se le llama anidación y permite crear programas donde se valoren expresiones complejas.
Sintaxis **if**(*expresión lógica*) { *sentencias;* } Si sólo se va a ejecutar una sentencia, no hace falta usar las llaves: **if**(*expresión lógica*) *sentencia*	Sintaxis: **if**(*expresión lógica*){ *sentencias;* } **else** { *sentencias;* } Las llaves son necesarias sólo si se ejecuta más de una sentencia.	Sintaxis y Ejemplo 3: **if** (*x==1*) { *//sentencias;* } **else** { **if**(*x==2*) { *//sentencias;* } **else** { **if**(*x==3*) { //sentencias; } }} Pero si cada else tiene dentro sólo una instrucción if entonces se podría escribir ***if-else-if***: **if** (*x==1*) { *// sentencias;* } **else if** (*x==2*) { *// sentencias;*} **else if** (*x==3*) { *// sentencias}*
Ejemplo 1: cout<<"Valor a?"; cin>>a; cout<<"Valor b?"; cin>>b; if(a>b){ cout<<"mayor: "; cout<<a; }	Ejemplo 2: cout<<"Valor a?"; cin>>a; cout<<"Valor b?"; cin>>b; if(a>b){ cout<<"mayor: "; cout<<a;} else { cout<<"mayor: "; cout<<b; }	

Estructura switch

Se trata de una sentencia que permite construir alternativas múltiples. Pero que en el lenguaje C/C++ está muy limitada. Sólo sirve para evaluar el valor de una variable entera (o de carácter, *char*).
Tras indicar la *expresión entera* que se evalúa, a continuación se compara con cada valor agrupado por una sentencia case. Cuando el programa encuentra un *case* que concuerda con el valor de la expresión se ejecutan las sentencias de todos los *case* siguientes. Por eso se utiliza la sentencias break para hacer que el programa abandone el bloque *switch. El default es opcional y, en el caso en que la expresión entera no concuerde con ningún valor de los case, se ejecutaran las sentencias que están a continuación.*

Sintaxis	Ejemplo
switch(*expresión_entera*)**{** **case** *valor1:sentencias;*break; */*Para que programa salte fuera del switch de otro modo atraviesa todos los demás case */* **case** *valor2:*sentencias;break; ... **default:***sentencias; break;* ***}***	switch (diasemana) { case 1:cout<<"Lunes";break; case 2:cout<<"Martes";break; case 3:cout<<"Miércoles";break; case 4:cout<<"Jueves";break; case 5:cout<<"Viernes";break; case 6:cout<<"Sabado";break; case 7:cout<<"Domingo";break; default:cout<<"Error";break; }

Trabajo Práctico

Los siguientes ejercicios deben resolverse aplicando el análisis y la Implementación en Código C++.

1) Ingresar un número por teclado, calcular y mostrar por pantalla su triple.
2) Ingresar 3 valores numéricos que corresponden a tres calificaciones escolares. Calcular y mostrar su promedio.
3) Calcular y mostrar por pantalla el volumen de una esfera a partir del ingreso por teclado de su radio.
   (La fórmula es $V=4/3.\pi.radio^2$)
4) Determinar y mostrar si un número ingresado por pantalla es par o impar.
5) Solicitar al usuario que ingrese dos números y mostrar cuál de los dos es menor.
6) Escriba un programa que pida dos números enteros y que escriba si el mayor es múltiplo del menor.
7) Desarrollar de un programa que permita leer una letra ingresada por el usuario que corresponda a un número romano y encontrar su valor en números decimales, en caso contrario decir no es un número romano válido.
8) Diseñar un programa que permita ingresar por teclado una letra. Luego, debe indicar por pantalla si es vocal o no. Y si lo fuera, escribir una palabra que comience con esa letra.

TEMA: SENTENCIAS DE REPETICION EN C++

Estructuras Repetitivas en general. Variables Contadoras y Acumuladoras

Las estructuras de control repetitivas, son aquellas que permiten ejecutar un conjunto de instrucciones varias veces, de acuerdo al valor que genere la expresión relacional y/o lógica. Esto significa que una instrucción repetitiva permite saltar a una instrucción anterior para volver a ejecutarla.
A las estas estructuras se les conoce también como ciclos o bucles, por su funcionamiento. Existen 3 estructuras repetitivas:

1. While
2. Do-while
3. For

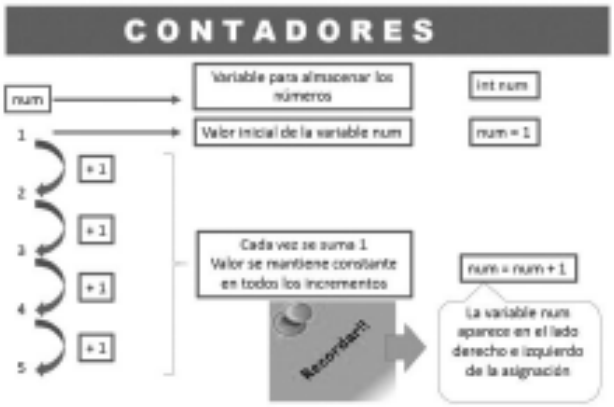
Las tres instrucciones tienen el mismo fin, y difieren únicamente en su sintaxis, siendo posible sustituir una solución en la que se utiliza "while", por una en la que se utiliza "do-while" o "for".
En los diagramas de flujo, un ciclo se representa de la siguiente manera:

En la imagen se puede observar que las líneas de flujo, indican el orden a seguir y según el valor de la condición, continuará ejecutándose el mismo conjunto de instrucciones o saldrá del ciclo. Entre las tres instrucciones hay pequeñas variaciones de representación gráfica que serán detalladas en la explicación de uso de cada una de ellas.
Las estructuras de control repetitivas utilizan dos tipos de variables: Contadores y Acumuladores.

VARIABLE CONTADORA

Un contador es una variable de tipo entero, que incrementa o decrementa su valor de forma **CONSTANTE** y requiere ser inicializada generalmente en 0 o 1, aunque en realidad depende del problema que se está resolviendo. Como su nombre lo indica se utilizan en la mayoría de veces para contar el número de veces que se ejecuta una acción, o para contar el número de veces que se cumple una condición (expresión relacional/lógica).
Los contadores se utilizan con la finalidad de contar sucesos o acciones internas de un bucle; deben realizar una operación de inicialización y posteriormente las sucesivas de incremento o decremento del mismo.
La inicialización consiste en asignarle al contador un valor inicial. Se situará antes y fuera del bucle.

> La variable que cumple el rol de contador, aparece tanto a la izquierda como a la derecha, por la propiedad destructiva de la asignación; así tomará el valor anterior, le adicionará o reducirá el valor constante y asignará el nuevo valor.



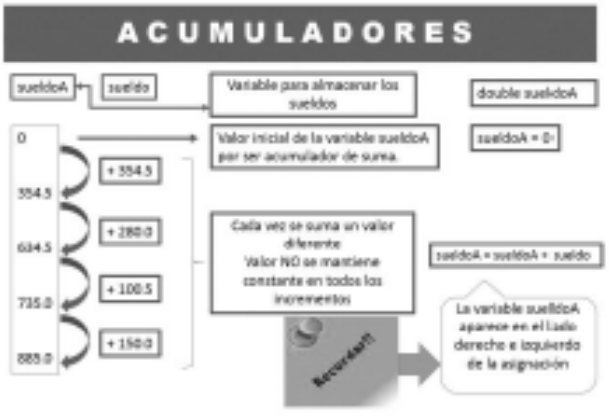
VARIABLE ACUMULADORA O SUMADORA

Un acumulador es una variable numérica, que incrementa o decrementa su valor de forma **NO CONSTANTE** y requiere ser inicializada. Como su nombre lo indica se utilizan para acumular valores en una sola variable, ya sea de suma o producto. Por lo tanto existen dos modos de inicialización:

- Para Suma: Inicializar en 0
- Para Producto: Inicializar en 1

Esto con el objetivo de no alterar los valores de las respectivas operaciones.
La diferencia entre un contador y un acumulador es que mientras el primero va aumentando de uno en uno, el acumulador va aumentando en una cantidad variable.

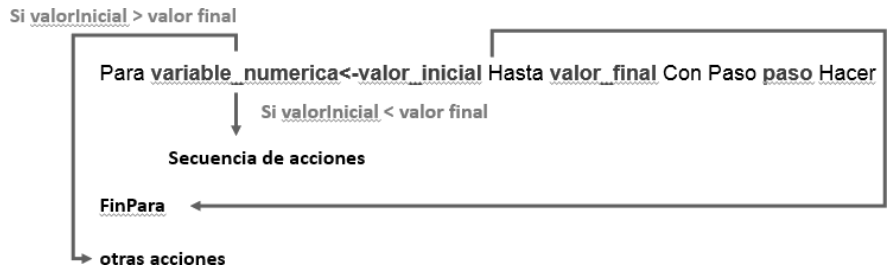
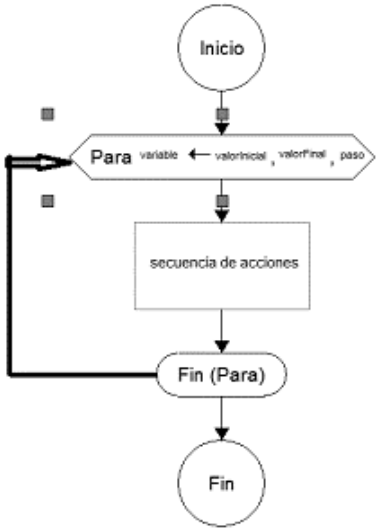
> La variable que cumple el rol de acumulador, aparece tanto a la izquierda como a la derecha, por la propiedad destructiva de la asignación; así tomará el valor anterior, le adicionará o reducirá el valor constante y asignará el nuevo valor.



Unidad 6.1 – Estructura for

La instrucción **"for"**, es una tercera estructura de control repetitiva, su principal característica radica en el hecho de que dentro de sí misma, constan la inicialización de variables, así como también las variables de incremento/decremento, necesarias en un ciclo. En este ciclo la secuencia de acciones se realiza mientras un valor inicial llega a un valor final.

DISEÑO

PSEUDOCÓDIGO	DFD
Para **variable_numerica<-valor_inicial** Hasta **valor_final** Con Paso **paso** Hacer secuencia_de_acciones Fin Para Se lee de la siguiente forma: Para la variable_numérica que empieza en valor_inicial, hasta que llegue al valor_final, incrementándose o decrementándose en el valor de "paso" haga la secuencia de acciones. Gráficamente se vería así:  	

Se trata de un bucle especialmente útil para utilizar contadores. Su formato es:

for(inicialización;condición;incremento){
  sentencias
}

Las sentencias se ejecutan mientras la condición sea verdadera. Además antes de entrar en el bucle se ejecuta la instrucción de inicialización y en cada vuelta se ejecuta el incremento. Es decir el funcionamiento es:

[1] Se ejecuta la instrucción de inicialización
[2] Se comprueba la condición
[3] Si la condición es ***verdadera***, entonces se ejecutan las sentencias. Si la condición es ***falsa***, abandona el bloque ***for***
[4] Tras ejecutar las sentencias, se ejecuta la instrucción de incremento y se vuelve al paso 2

Ejemplo : mostrar números del 1 al 1000
Para resolver este ejercicio se debe pensar el ciclo for

for(inicialización;condición;incremento){
variable++ (incrementa en 1 el valor de la variable en cada vuelta)
variable<=cantidad de veces que se repite
variable=valor_inicial;

```
for (int i=1;i<=1000;i++){
cout<<i<<" ";
}
```

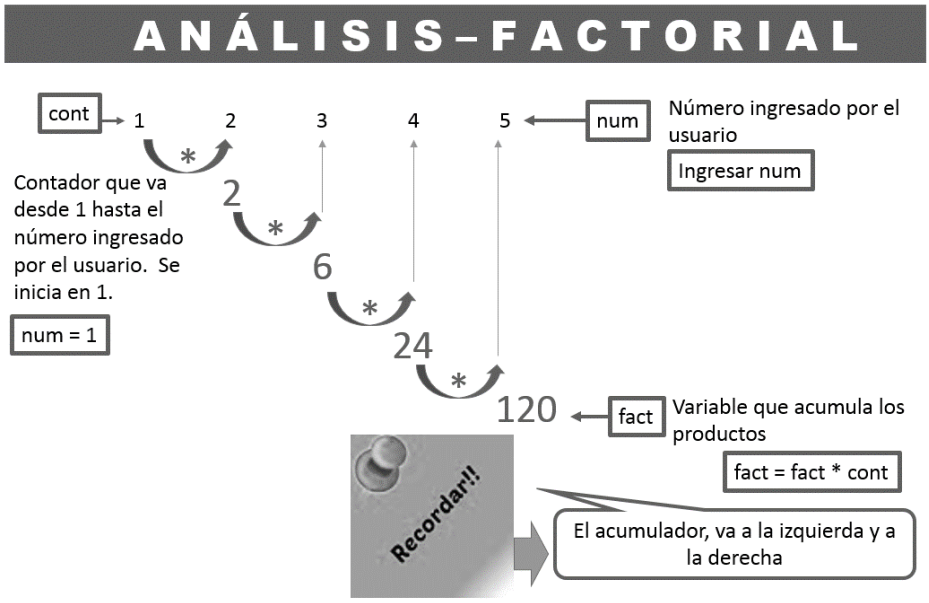
La variable **i** que controla el ciclo for debe ser del tipo entero (int) y puede declararse antes de la sentencia o dentro del ciclo (como en el ejemplo). Dicha variable toma los valores de 1 a 1000 según cada vuelta de ciclo.

EJEMPLO: for

Realice un programa que permita obtener el factorial de un número dado por el usuario, utilizando la instrucción "for".

ANÁLISIS

Paso1: Comprender lo que se pide: El **factorial** de un entero positivo n, se define como el producto de todos los números enteros positivos desde 1 hasta n.
Matemáticamente 5! = 5*4*3*2*1 ó a la inversa: 5! = 1*2*3*4*5
Una vez conocido el proceso matemático, es conveniente hacer una representación gráfica de todas las variables que intervienen.

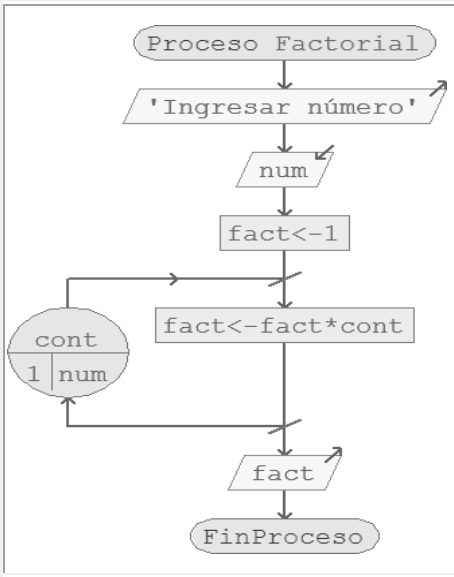


Así en la gráfica puede observarse es necesario ir incrementando el valor en 1 (valor constante - CONTADOR) para llegar hasta el 5 (número dado por el usuario); es decir, 1*2 = 2, este resultado se multiplica por 3 (2*3 = 6), 6 se multiplica por 4 (6*4=24) y, 24 se multiplica por 5 que es el número del cual se quiere obtener el factorial (24 * 5 = 120) por lo que es posible observar como los resultados de las multiplicaciones se acumulan hasta obtener el factorial.
***Atención**: las variables acumuladoras pueden acumular sumando o multiplicando.*
Paso 2: ¿Qué datos necesito para solucionar el problema?
Entradas:
num // variable del tipo entero
Paso 3: ¿Qué resultados se quieren obtener?
Salidas:
fact // variable del tipo entero
Paso 4: ¿se necesitan almacenar otros datos en variables para realizar lo pedido? Si es así, ¿cuáles y de qué tipo?
cont // variable contadora del tipo entero

DISEÑO: Pseudocódigo

```
Proceso Factorial
Escribir 'Ingresar número'
Leer num                    //Dato ingresado por el usuario
                            //La instrucción "for" permite inicializar dentro de su propia estructura.
                            //Las inicializaciones se realizan UNA SOLA VEZ
                            //La instrucción "for" también permite realizar incrementos y decrementos dentro de su propia
                            // estructura. Se realizan VARIAS VECES
fact<-1
for (cont<-0, fact<-1 ; cont<=num; cont++)
    fact<-fact*num //Instrucción de acumulador, se realiza VARIAS VECES, va dentro del ciclo
FinFor
Escribir fact
FinProceso
```

DISEÑO: DFD



El diseño toma la mayoría de datos de la fase de análisis, para incorporarlo a un proceso secuencial. Al distinguir cuáles son las acciones que se realizan una sola vez y las que se realizan varias veces, éstas últimas son aquellas que van dentro de las instrucciones repetitivas.

IMPLEMENTACIÓN en C++

```
#include<iostream>
using namespace std;
main()
{
int num, cont, fact;

cout<<”Ingrese número”;                  // Imprime o muestra un mensaje por pantalla
cin>>num;                                // ingresa número y lo guarda en la variable
fact=1;                                  //Inicializa acumulador
for(cont=0; cont<=num;cont++){
   fact = fact * cont;                   //Instrucción de acumulación
}
cout<<"Factorial: "<<fact;
}
```

Trabajo Práctico Estructura for

Actividad 1. *Analizar el código de cada ítem e indicar que se muestra por pantalla:*

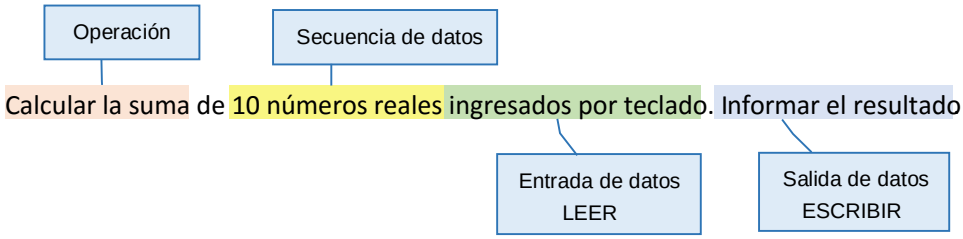
a)
```
for(i=0;i<10;i++){
    cout<<i*2;
```
b)
```
for(i=0;i<12;i++){
    cout<<”hola”;endl;
```
c)
```
for(i=0;i>10;i++){
    cout<<i*10;
```
d)
```
for(int i=1;i<=50;i++){
        if(i%2==0){cout<<i;}
}
```
e)
```
for(i=0;i<3;i++){
    cout<<”*”;
    for(j=0;j<5;j++){
        cout<<”+”;
    }
}
```

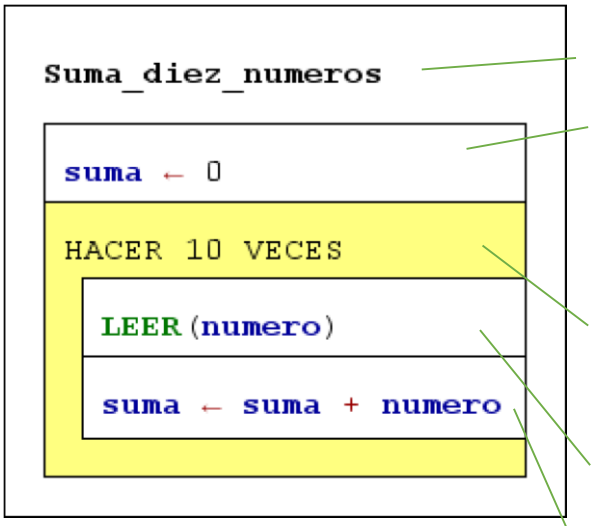
Actividad 2. Ejercicios con guía de análisis

Los siguientes ejercicios deben resolverse realizando código en C++ aplicando el análisis detallado para cada caso

1. Calcular la suma de 10 números reales ingresados por teclado. Informar el resultado.

ANÁLISIS DEL PROBLEMA





Nombre del programa

Inicializa la variable *suma* que será utilizada en la operación de acumulación. De lo contrario daría lugar a error. Aclaración: Inicializar no es declarar (o definir) la variable para el compilador. Aquí sólo asignamos los valores mínimos que el algoritmo necesita para trabajar sin errores. Por ejemplo, no se inicializa *numero*, ya que su primer valor será leído desde teclado.

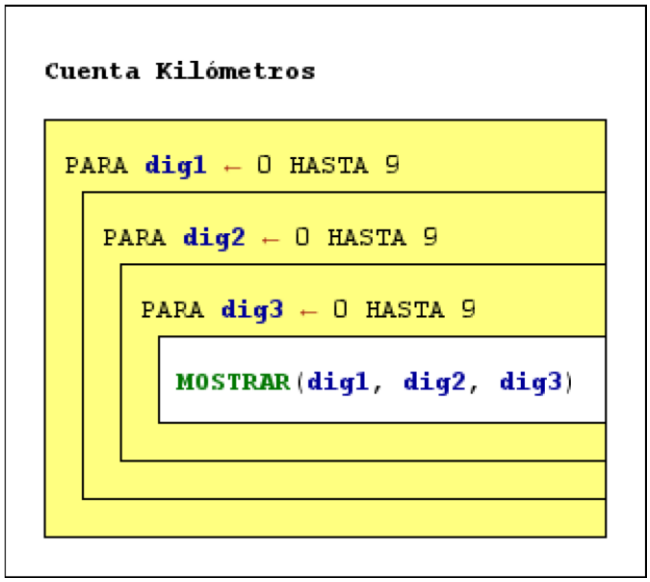
Se repetirán 10 veces las acciones contenidas en este bloque. HACER no necesita manipular una variable extra para contar la cantidad de iteraciones, ya que el número de iteraciones es conocido

La palabra reservada LEER deja en espera el procesador hasta que el usuario escribe el número, luego lo guarda en la variable *numero*

La variable suma acumula en cada pasada el contenido del nuevo número, sumándolo al anterior. En la última pasada habrá acumulado los 10 valores. No se pide

2. Representar el movimiento de un cuenta kilómetros desde 000 a 999.

ANALISIS DEL PROBLEMA



El primer dígito varía de 0 a 9

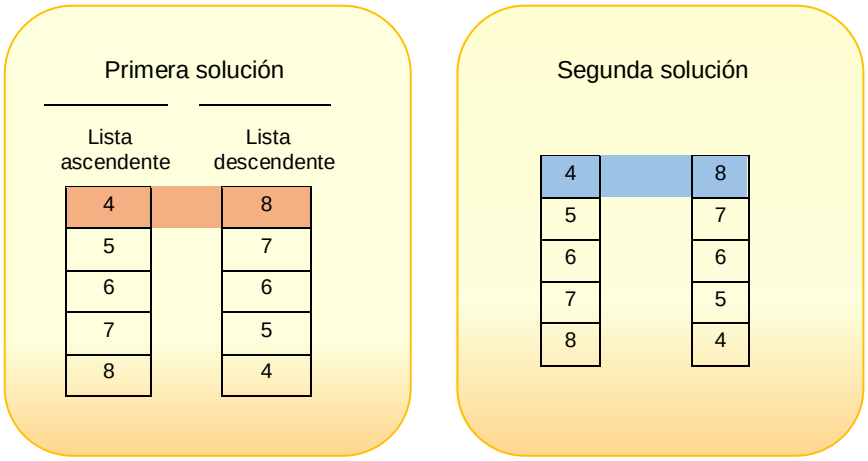
El segundo dígito varía de 0 a 9

El tercero dígito varía de 0 a 9. En principio todos muestran 0. Cuando el dígito 3 de las unidades avance hasta 9, se incrementará en 1 el dígito 2 de las decenas. Sólo cuando el dígito 2 avance hasta 9, se incrementará en 1 el dígito 3 de las centenas.

3. Ingresar dos números e informar los números naturales comprendidos entre ellos en orden ascendente y descendente utilizando un solo ciclo.

ANÁLISIS DEL PROBLEMA

La secuencia de datos es un intervalo abierto (a;b), es decir, excluyendo de la sucesión a estos valores. Se pide diferenciar entre orden ascendente y descendente, lo cual implica averiguar cuál de los números es menor que el otro. Una segunda solución más simple los muestra en pantalla como dos listas invertidas desconociendo el procesador cuál es la ascendente y cuál la descendente.



Actividad 3.

Los siguientes ejercicios deben resolverse aplicando el análisis, el diseño (en pseudocódigo y Diagrama de Flujo) y la Implementación en Código C++. Para subir la resolución de los mismos puede incluirse todo en un archivo de texto.

Ejercicio 1:

Crear un programa que muestre en pantalla los números pares del 26 al 10.

Ejercicio 2:

Crear un programa que muestre en pantalla el cuadrado de los primeros 20 números enteros positivos.

Ejercicio 3:

Crear un programa que para un total de 7 alumnos ingresar nombre y promedio. Si el promedio es mayor o igual a 7 mostrar un cartel que diga APROBADO, sino DESAPROBADO. Calcular y mostrar promedio máximo y promedio mínimo.

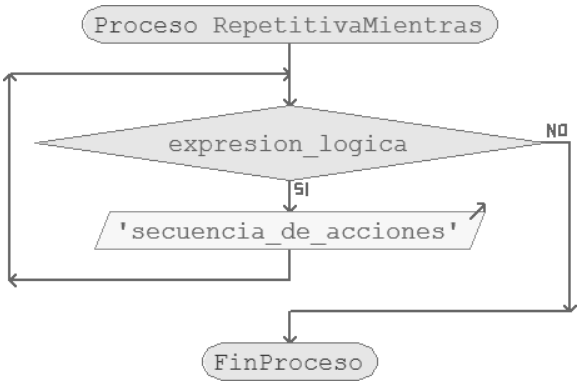
Ejercicio 4:

Ingresar un número entero por teclado y mostrar su tabla de multiplicar. Repetirlo para 5 números.
(Pista: for dentro de otro for -> for anidados. Mirar ejemplos y ejercicios anteriores)

Estructura while

La instrucción **"While"**, es una estructura de control repetitiva que puede impedir la ejecución de un conjunto de instrucciones, si la evaluación de la expresión relacional y/o lógica es falsa. Esto significa que se convierte en repetitiva únicamente cuando la evaluación de la condición es verdadera.

DISEÑO

PSEUDOCÓDIGO	DFD
Mientras (**expresión_lógica**) secuencia_de_acciones **Fin Mientras** Se lee de la siguiente forma: Mientras la evaluación de la expresión lógica sea verdadera realice la secuencia de acciones. Esto significa que si la primera vez, que evalúa la expresión lógica, obtiene un valor "false", entonces la secuencia de acciones no se realizaría nunca, pero si es verdadera "true" entonces la secuencia de acciones se realizaría varias veces hasta que el valor de la expresión lógica sea falso. Al llegar a Fin Mientras siempre volverá automáticamente a la expresión lógica. Gráficamente se vería así:  	

IMPLEMENTACION EN C++

Es una de las sentencias fundamentales para poder programar. Se trata de una serie de instrucciones que se ejecutan continuamente mientras una expresión lógica sea cierta.

Sintaxis:

while (*expresión lógica*) { **//la variable de la expresión lógica recibe el nombre de centinela o**
sentencias… **//bandera ya que de su valor depende el corte del ciclo**
}

El programa se ejecuta siguiendo estos pasos:

[1] Se evalúa la ***expresión lógica***

[2] Si la expresión es verdadera ejecuta las **sentencias**, sino el programa abandona la sentencia **while**

[3] Tras ejecutar las sentencias, vuelve al paso 1

Ejemplo: mostrar números del 1 al 100

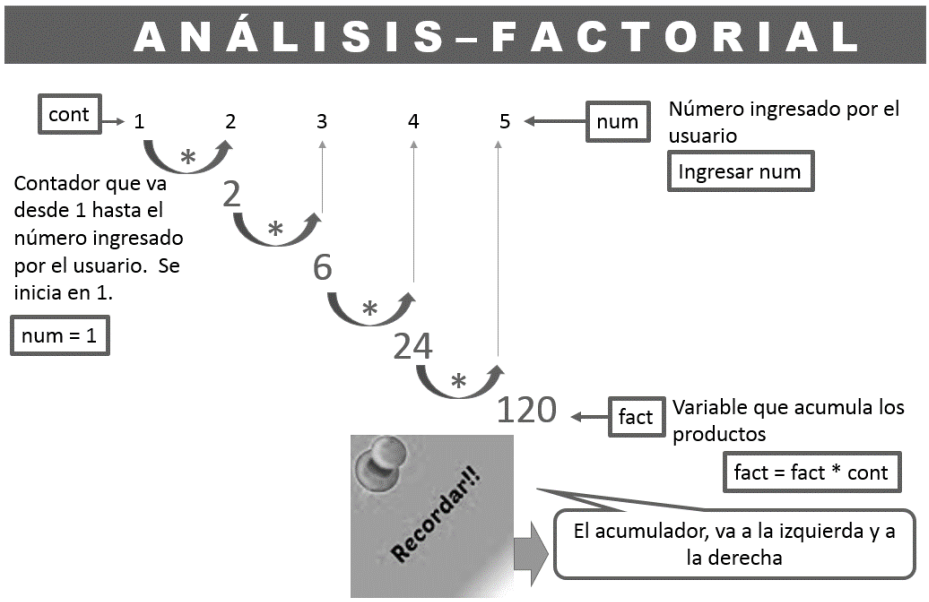
```
int i=1;
while (i<=100){     //evalúa la expresión lógica si es verdadera ejecuta las sentencias; si es falsa, termina el ciclo
std::cout<<i<<" "; // muestra el valor de i
i++;                //incrementa en 1 a la variable i -> variable contadora (ver apartado)
}while (i<=100){    //evalúa la expresión lógica si es verdadera ejecuta las sentencias; si es falsa, termina el ciclo
std::cout<<i<<" "; // muestra el valor de i
i++;                //incrementa en 1 a la variable i -> variable contadora (ver apartado)
}
```

EJEMPLO: while

Realice un programa que permita obtener y mostrar por pantalla el factorial de un número dado por el usuario usando la sentencia while

ANÁLISIS

Paso1: Comprender lo que se pide: El **factorial** de un entero positivo n, se define como el producto de todos los números enteros positivos desde 1 hasta n.
Matemáticamente 5! = 5*4*3*2*1 ó a la inversa: 5! = 1*2*3*4*5
Una vez conocido el proceso matemático, es conveniente hacer una representación gráfica de todas las variables que intervienen.

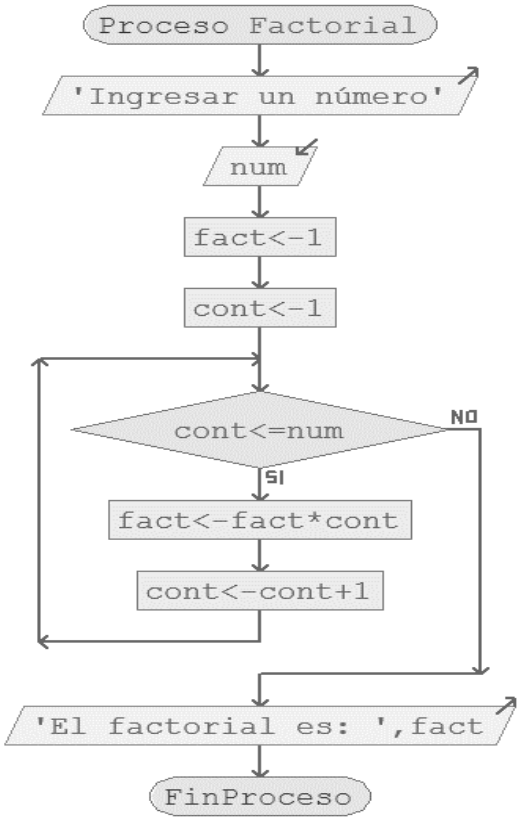


Así en la gráfica puede observarse es necesario ir incrementando el valor en 1 (valor constante - CONTADOR) para llegar hasta el 5 (número dado por el usuario); es decir, 1*2 = 2, este resultado se multiplica por 3 (2*3 = 6), 6 se multiplica por 4 (6*4=24) y, 24 se multiplica por 5 que es el número del cual se quiere obtener el factorial (24 * 5 = 120) por lo que es posible observar como los resultados de las multiplicaciones se acumulan hasta obtener el factorial.
***Atención**: las variables acumuladoras pueden acumular sumando o multiplicando.*
Paso 2: ¿Qué datos necesito para solucionar el problema?
Entradas:
num // variable del tipo entero
Paso 3: ¿Qué resultados se quieren obtener?
Salidas:
fact // variable del tipo entero
Paso 4: ¿se necesitan almacenar otros datos en variables para realizar lo pedido? Si es así, ¿cuáles y de qué tipo?
cont // variable contadora del tipo entero

DISEÑO: Pseudocódigo

```
Proceso Factorial
Escribir 'Ingresar número'
Leer num              //Dato ingresado por el usuario
cont<-0               //Inicializa el contador, se realiza una SOLA VEZ, va fuera del ciclo
fact<-1               //Inicializa el acumulador, se realiza una SOLA VEZ, va fuera del ciclo
Mientras cont<=num Hacer
 fact<-fact*num       //Instrucción de acumulador, se realiza VARIAS VECES, por eso va en el ciclo
 cont<-cont+1         //Incremento constante del contador, se realiza VARIAS VECES, por eso va en el ciclo
FinMientras
Escribir fact
FinProceso
```

DISEÑO: DFD



El diseño toma la mayoría de datos de la fase de análisis, para incorporarlo a un proceso secuencial. Al distinguir cuáles son las acciones que se realizan una sola vez y las que se realizan varias veces, éstas últimas son aquellas que van dentro de las instrucciones repetitivas.

IMPLEMENTACIÓN en C++

```
#include<iostream>
using namespace std;
main()
{
int num, cont, fact;
string mesLetras;

cout<<”Ingrese número”;                 // Imprime o muestra un mensaje por pantalla
cin>>num;                               // ingresa número y lo guarda en la variable

fact=1;                                 //Inicializa acumulador
cont=1;                                 //Inicializa contador
while(cont <= num){
   fact = fact * cont;                  //Instrucción de acumulación
   cont = cont + 1;                     //Instrucción de contador
}                                       // fin while
cout<<"Factorial: "<<fact;
}
```

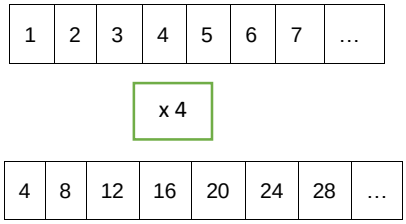
Trabajo Teórico Práctico – Estructura while

Actividad. *A partir del análisis del problema resolver los siguientes ejercicios a través de lenguajes algorítmicos: pseudocódigo DFD. Luego, realizará la implementación en código C++ aplicando el análisis detallado para cada caso. Recordar que*
ANÁLISIS DEL PROBLEMA

a) ***Datos de Entrada***
b) ***Datos de Salida***
c) ***Proceso***

EJERCICIO 1. Calcular y mostrar en forma ascendente los múltiplos de 4 menores que n.

La secuencia de múltiplos de 4 se obtiene multiplicando 4 por cada uno de los números naturales hasta llegar a un múltiplo cercano a 60.



Si bien el programador conoce la secuencia de números, se pide que el programa los calcule de a uno y los vaya mostrando en forma creciente, suponiendo que se desconoce el último. La condición será que el múltiplo obtenido sea menor que 60.

EJERCICIO 2. Desarrollar código en C++ que permita al usuario Ingresar por teclado cierta cantidad de números (la que el usuario desee) y luego debe mostrar por pantalla la cantidad de números positivos y negativos por separado.
(Ayuda: usar ciclo repetitivo y estructura de decisión)

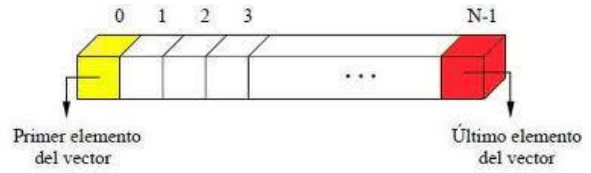
EJERCICIO 3. Diseñar un programa que registrar cada venta de un negocio. Al final del día, debe mostrar la cantidad total de ventas realizadas, la suma total de ventas y el promedio de las mismas. Además, deberá mostrar venta mínima y venta máxima.

EJERCICIO 4: Crear un programa que permita a un usuario ingresar tantas veces como lo desee el nombre de un alumno y su calificación. La cantidad de alumnos no se conoce de antemano. Cuando el usuario así lo desee, el programa debe permitirle terminar el ingreso de datos. A continuación, le mostrará la cantidad de alumnos aprobados (calificación >=7) y los desaprobados (calificación <7). Finalmente, debe mostrar el promedio de notas de todos los alumnos.

TEMA: ARREGLOS UNIDIMENSIONALES EN C++

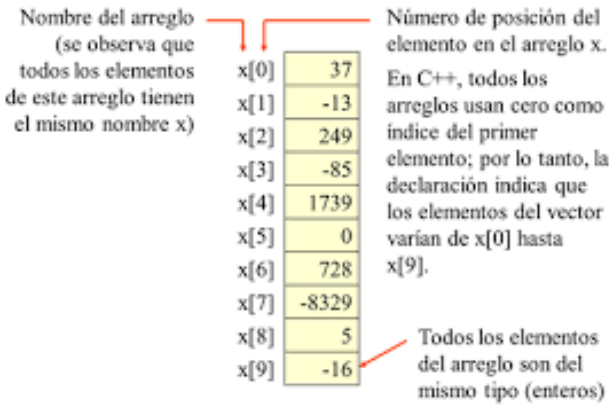
Los arreglos pueden definirse como una colección de datos del mismo tipo que se denominan o referencian por un mismo nombre y que son almacenados en posiciones de memoria contiguas. Un arreglo es, en resumen, un conjunto de datos finito y del mismo tipo.

ARRAYS UNIDIMENSIONALES

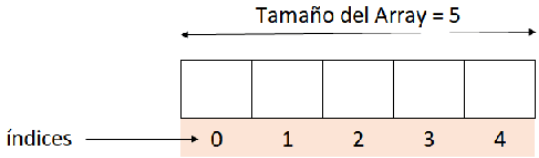


Con esta imagen es posible observar que un array (arreglo) unidimensional es una estructura, dividida en celdas, donde cada una de ellas representa un espacio de almacenamiento. Para acceder a sus posiciones se utilizan los pequeños números que en la figura aparecen la parte superior, se les denomina índices, y representan las posiciones. El primer índice en el caso del lenguaje de programación C++ se inicia en 0.
Por lo tanto, un array (unidimensional, también llamado arreglo o vector) es una variable estructurada formada por un número ***"n"*** de variables simples del mismo tipo que son los componentes o elementos del array.
El número de componentes ***"n"*** es la dimensión o tamaño del array. Cada elemento de un array unidimensional se identifica con un índice, también llamado posición.

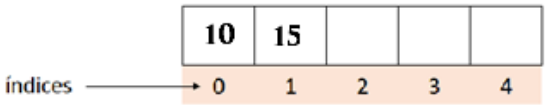
Ejemplo:



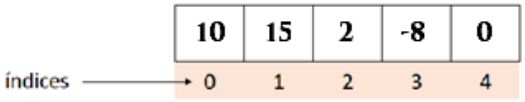
Declaracion de un Arreglo: se debe especificar: tipo, nombre y cantidad de posiciones de memoria
Tipo_Dato Nombre_Array[cantidad];
int A[5]; // Arreglo cuyo nombre es A de 5 elementos que almacena tipo de datos enteros



Asignación: Se debe especificar nombre y posición.
A[0]=10; // En la posicion 0 de A se asigna el valor entero 10
A[1]=B+8; // sabiendo que B=7, en la posicion 1 de A se asigna el valor entero resultante de la suma



Inicializacion: Se le asignan valores iniciales al arreglo entre llaves.
int A[5]={10,15,2,-8,0};



Escritura y Lectura: Se especifica nombre del Array y posicion usando las sentencias correspondientes.

- *cout<<A[3];* // muestra -8, que es el contenido de A en la posicion 3
- *cin>>A[4];* // asigna el valor ingresado por teclado en la posicion 4 de A

Ejemplo 1:

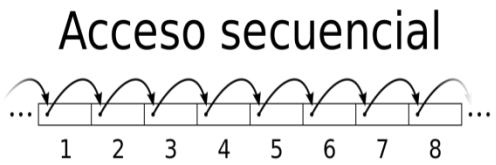
```
#include <iostream>
using namespace std;
int main ()
{
int n, m = 5;
int A[]={10,15,2,-8,0}; //se declara un array de 5 elementos
                        //de tipo entero con sus valores iniciales
 n = A[2];      //A n se le asigna el valor contenido en la
                // posición 0 del Arreglo A; o sea, n= 2
A[0]=A[1]+A[2]; //En la posición 0 del Arreglo A se guarda la
                //suma del valor contenido en la posición 1 y
                // el de la posición 2; o sea, A[0]= 15 + 2 = 17
A[1]++; //le suma 1 al valor contenido en la posición 1
        //osea A[1]= 16
A[n]=m+10;  //En la posición n (n es 2) del Arreglo A se
            //guarda la suma de m (que es 5) y 10; o sea,
            //A[2]= 15
A[n+1]=7;   //En la posición n+1 (n que es 2 se le suma 1) del
            //Arreglo A se guarda el valor 7; o sea, A[3]= 7
if(A[0]>=A[1])  // Si el valor de la posición 0 del arreglo A es
                //mayor o igual al valor de la posición 1 del
                //arreglo A (o sea, if(17 >= 9) entonces…
    A[4]=A[0]; //A la posición 4 del Arreglo A se le asigna el
               // valor contenido en la posición 0; sea,
               // A[4]= 17
cout<<A[0]<<" "<<A[1]<<" "<<A[2]<<" "<<A[3]<<" "<<A[4];
//se muestra los valores almacenados en el Arreglo A
cout << endl;

}
```

Al ejecutar el programa, la pantalla mostrará:
17 16 2 7 17

Sobre un arreglo es posible realizar una variedad de operaciones, las cuales permiten manipular el arreglo de acuerdo a las necesidades que se presenten al administrar información con este tipo de estructura:
Para facilitar dichas operaciones es necesaria acceder de forma secuencial a cada posición del arreglo.

ARRAYS UNIDIMENSIONALES – Acceso Secuencial

Para acceder a cada posición del arreglo se recorren, de a 1, todas las posiciones del mismo.



Ejemplo: Si declara el siguiente arreglo:
Int B[4];

Arreglo B →

0	1	2	3

Y se quiere completar ingresar por teclado los valores de cada posición o índice secuencialmente .
Se indican los pasos:

- Primer paso:Se ingresa por teclado el valor 5 a la posición 0 del arreglo
En C++, se escribe: cin>>B[0]=5;

Arreglo B →

5			
0	1	2	3

- Segundo paso: Se ingresa por teclado el valor 8 a la posición 1 del arreglo
En C++, se escribe: cin>>B[1]=5;

Arreglo B →

5	**8**		
0	1	2	3

- Tercer paso: Se ingresa por teclado el valor 0 a la posición 2 del arreglo
En C++, se escribe: cin>>B[2]=0;

Arreglo B →

5	**8**	**0**	
0	1	2	3

- Cuarto paso: Se ingresa por teclado el valor 2 a la posición 3 del arreglo
En C++, se escribe: cin>>B[3]=2;

Arreglo B →

5	**8**	**0**	**2**
0	1	2	3

Observemos y analisemos…
- En el primer paso, se ubica en la primer posición del arreglo cuyo indice es igual a 0.

Arreglo B →

5			
0	1	2	3

- En el segundo paso, se ubica en la segunda posición del arreglo cuyo indice es igual a 1.

Arreglo B →

5	**8**		
0	1	2	3

Osea, que el índice que antes era 0 ahora toma el valor 1. Uno mas que el anterior.

- En el tercer paso, se ubica en la tercera posición del arreglo cuyo indice es igual a 2.

Arreglo B →

5	**8**	**0**	
0	1	2	3

Osea, que el índice que antes era 1 ahora toma el valor 2. Uno mas que el anterior.

- En el cuarto paso, se ubica en la cuarta posición del arreglo cuyo indice es igual a 3.

Arreglo B →

5	**8**	**0**	**2**
0	1	2	3

Osea, que el índice que antes era 2 ahora toma el valor 3. Uno mas que el anterior.

En conclusión, el indice se inicia en 0 y se incrementa de a 1 hasta completar el arreglo cuya cantidad de elementos es 4.

Pensemos y recordemos…
- En el acceso secuencial, hay acciones que se repiten.
- En el acceso secuencial, existe un valor (el índice del arreglo) que se incrementa de uno en uno.

¿Qué estructura de repetición realiza cierta cantidad de acciones pudiendo controlar la cantidad de veces en que se repiten incrementando en uno la variable que controla dichas repeticiones?...
… La estructura de repetición for

¿Qué datos del arreglo se necesitan para este ejemplo?
- Nombre del arreglo
- Cantidad de elementos
- índice del arreglo

El nombre del arreglo es B, la cantidad de elementos que contiene es 4 y el índice del arreglo será la variable de control del ciclo for a la que llamaremos i.

```
for (int i=0; i<4;i++){
  cin>> B[i]=valor_entero_que_se_ingresa;
}
```

1º vuelta del ciclo → **i = 0**

5			
0	1	2	3

2º vuelta del ciclo → **i = 1**

5	**8**		
0	1	2	3

3º vuelta del ciclo → **i = 2**

5	**8**	**0**	
0	1	2	3

4º vuelta del ciclo → **i = 3**

5	**8**	**0**	**2**
0	1	2	3

SINTAXIS DE OPERACIONES CON ACCESO SENCUENCIAL EN ARREGLOS UNIDIMENSIONALES

Lectura o ingresar datos a un arreglo:

```
for (int i=0; i<cantidad_elementos;i++){
  cin<< A[i];
}
```

Escritura o Mostrar datos contenidos en el arreglo:

```
for (int i=0; i<cantidad_elementos;i++){
  cout<< A[i];
}
```

Asignación de datos en cada posicion del arreglo:

```
for (int i=0; i<cantidad_elementos;i++){
   A[i]=valor_que_se_asigna;
}
```

Suma de los elementos de un arreglo.

Recordemos…

¿Cómo se sumaban 10 valores ingresados por teclado usando la estructura de repetición for?

```
#include<iostream>
using namespace std;
int main()
{
 int numero;
 int suma=0; // variable para guardar la suma. Como es
             // acumuladora la inicializo en cero.

 for (i=0;i<10;i++)
 {
          cout<<"ingresar un numero";
          cin>>numero;
          suma=suma+numero; // se guarda en suma
                                //cada número ingresado
 } // fin del for…
 cout<<"La suma es "<<suma<<endl;
}
```

Ahora, en lugar de usar una sola varible usaremos un conjunto de variables que forman parte de un arreglo. Veamos como:

```
#include<iostream>
using namespace std;
int main()
{
 int numeros [10]; // declaro un arreglo para almacenar 10
                   // elementos del tipo de dato entero
 int suma=0; // variable para guardar la suma. Como es
             // acumuladora la inicializo en cero.
 for (i=0;i<10;i++)
 {
          cout<<"ingresar un numero";
          cin>>numeros[i]; // cada valor ingresado se guarda
                           //en el arreglo
          suma=suma+numeros[i]; // se guarda en suma
                                //cada número ingresado
 } // fin del for…
 cout<<"La suma es "<<suma<<endl;
}
```

Promedio, Valor Máximo y Valor Mínimo del conjunto de elementos de un arreglo unidimensional.

Ejemplo.

Ingresar por teclado y almacenar en un arreglo o vector 20 valores reales comprendidos entre 1 y 100.
A. Calcular y mostrar por pantalla el promedio de los valores del arreglo.
B. Buscar y mostrar por pantalla el máximo y el mínimo, junto a la posición de cada uno.

```
#include<iostream>
using namespace std;
int main()
{
 int numeros [20]; // declaro un arreglo para almacenar 20
                   // elementos del tipo de dato entero
 float prom; // variable para guardar promedio
 int suma; // variable para guardar la suma. Como es
           // acumuladora la inicializo en cero.
 mayor=0; //variable para guardar el número máximo
 menor=101; // variable para guardar el número mínimo

 for (i=0;i<20;i++)
 {
          cout<<"ingresar un numero";
          cin>>numeros[i]; // cada valor ingresado se guarda
                           //en el arreglo
          suma=suma+numeros[i]; // se guarda en suma
                                //cada número ingresado
          if(numeros[i]>mayor){
                   mayor=numero[i];} //guarda el máximo
          if(numeros[i]<menor){
                   menor=numero[i];} //guarda el mínimo
} // fin del for…
 prom=suma/20; // calcula promedio
 cout<<"El promedio es "<<prom<<endl;
 cout<<"el maximo es " <<mayor<<endl;
 cout<<"el maximo es " <<menor<<endl;
}
```

2. A) ACTIVIDADES TEÓRICAS

Ejercicio 2.A.1.- *Suponiendo que quisiéramos calcular la suma de los elementos, el valor máximo y mínimo del arreglo B usado en el ejemplo dado, analice e indique los valores que van tomando las variables que guardan la suma, el máximo y el mínimo en cada vuelta del ciclo for, sabiendo que sus valores de inicio son:*
Suma=0
Máximo=-1
Mínimo=11

1º vuelta del ciclo → **i = 0**

5			
0	1	2	3

Suma=¿?.....
Máximo=¿?.....
Mínimo=¿?....

2º vuelta del ciclo → | 5 | 8 | | |
i = 1 | 0 | 1 | 2 | 3 |

Suma=¿?.....
Máximo=¿?.....
Mínimo=¿?....

3º vuelta del ciclo → | 5 | 8 | 0 | |
i = 2 | 0 | 1 | 2 | 3 |

Suma=¿?.....
Máximo=¿?.....
Mínimo=¿?....

4º vuelta del ciclo → | 5 | 8 | 0 | 2 |
i = 3 | 0 | 1 | 2 | 3 |

Suma=¿?.....
Máximo=¿?.....
Mínimo=¿?....

Ejercicio 2.A.2.-
Se pide (a) Ingresar por teclado la nota final de cada uno de los 30 alumnos de un curso y almacenarlos en un arreglo. (b) Calcular y mostrar el promedio de notas finales de todos los alumnos. (Se debe sumar cada elemento del arreglo). (c) Calcular y mostrar cantidad de alumnos aprobados (nota>=7). (d) Calcular y mostrar cantidad de alumnos desaprobados (nota<7). Completar las líneas de código punteadas del siguiente programa para que pueda cumplir con lo solicitado.

```
#include<iostream>
using namespace std;
int main()
{
 float notas [30];
 float prom, suma=0;;
 int aprobados=0, desaprobados=0;

  for (i=0;i<30;i++)
  {
          cout<<"ingresar un nota";
          cin>> ……..
          suma = ……..
          if( ……. ){
                  aprobado = aprobado+1;}
          if( ……. ){
                  desaprobado = desaprobado+1;}
}
  prom= ……. ;
  cout<<"El promedio es "<<prom<<endl;
  cout<<"Alumnos aprobados:  " <<aprobados<<endl;
  cout<<"Alumnos desaprobados:  " <<desaprobados<<endl;
}
```

Ejercicio 2.A.3.-
Se pide ingresar por teclado y almacenar en un arreglo el precio de costo de 100 productos. Luego, se debe calcular el precio de venta almacenándolo en otro arreglo sabiendo que: Precio venta = Precio costo + 35% precio de costo. Finalmente, debe mostrar el precio de costo y el precio de venta de cada producto.
Completar las líneas de código punteadas del siguiente programa para cumplir con lo solicitado:

```
#include<iostream>
using namespace std;
int main()
{
 float PCOSTO [100], PVENTA[100];;
  for (i=0;i<100;i++)
  {
          cout<<"ingresar un Precio de Costo";
          cin>> ……..
          PVENTA[i]= …… + (0.35* ……..);
}
for (i=0;i<100;i++)
  {
cout<<"El precio de costo del producto es: "<< ……;
cout<<"El precio de venta del producto es: "<< ……;
}
}
```

2. B) ACTIVIDADES PRÁCTICAS

Los siguientes ejercicios pueden ser resueltos en computadora o celular usando compilador o la aplicación que corresponda, o en hoja de carpeta.
En ambos caso se debe desarrollar código en C++.
Para resolver los ejercicios, se sugiere:
- leer y analizar la teoría y los ejemplos detallados anteriormente.
- En la medida de lo posible, usar como soporte los siguientes videos propuestos:

- Suma de elementos de un arreglo https://www.youtube.com/watch?v=_JmyF2JPqwk
- Mostrar los elementos de un arreglo con sus índices https://www.youtube.com/watch?v=R12v-66sKJY
- Mayor elemento de un arreglo https://www.youtube.com/watch?v=APjGtdNqAbk
- Almacenando el contenido de dos arreglos en uno solo https://www.youtube.com/watch?v=6t9BRH8NCGQ

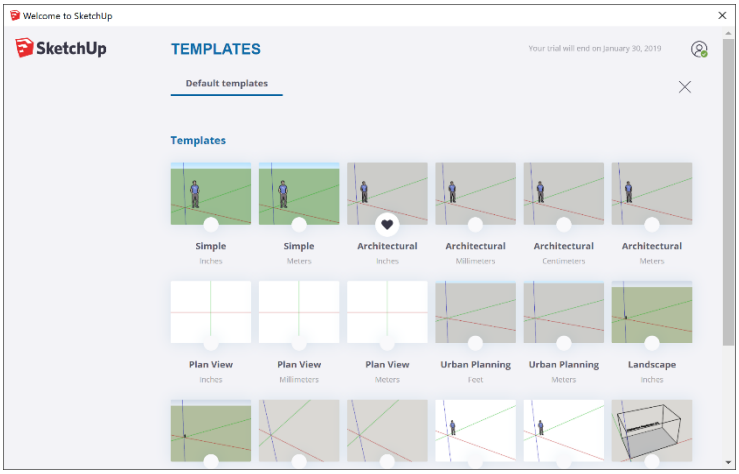
Ejercicio 2.B.1- Se tienen los COSTOS de producción de una fábrica correspondientes a los 12 meses anteriores al mes en curso. Elaborar un programa que, en un arreglo se guarde dicha información y que luego, con la implementación de funciones, pueda proporcionar la siguiente información
A. ¿En qué mes se registró el mayor costo de producción?
B. ¿Cuál es el promedio anual de costo de producción?
C. Cantidad de meses en que el costo de producción no supero los $100000.-

Ejercicio 2.B.2.- Realizar un programa que permita gestionar la información de medicamentos de una farmacia, teniendo en cuenta las tareas requeridas.
A. Ingresar datos de los medicamentos en 2 arreglos: CODIGO Y PRECIO.
B.- Mostrar Lista de Productos indicando Código y Precio de cada uno por pantalla.

TEMA: INTRODUCCIÓN A SKETCHUP

La primera vez que uses SketchUp, tendrás que iniciar sesión para activar tu suscripción o tu versión de prueba. Cuando hayas iniciado sesión, aparecerá la ventana de bienvenida de SketchUp, como se muestra aquí. Esta ventana es tu punto de partida para crear modelos, y aparece cada vez que abras SketchUp (a no ser que la desactives en la ventana de Preferencias de SketchUp).
En la ventana de bienvenida de SketchUp puedes seleccionar una plantilla para tu modelo, definir la plantilla por defecto, abrir modelos recientes, buscar un archivo existente, registrar una copia de SketchUp Pro (consulta Entender tu licencia para más detalles) o ver más información sobre SketchUp.

1. Seleccionar una plantilla

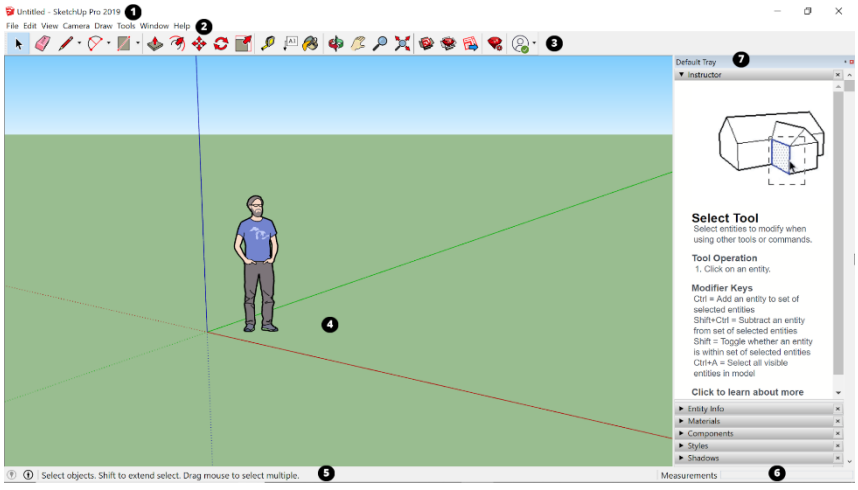


Todos los modelos de SketchUp se basan en una plantilla que tiene ciertos ajustes predeterminados sobre el fondo de tu modelo y sus unidades de medida. Cuando empiezas un modelo nuevo, seleccionar la plantilla con la unidad de medida adecuada te hace el modelado más fácil.
Así se selecciona una plantilla en la ventana de bienvenida de SketchUp:
1. En el panel de Archivos, que viene preseleccionado en la barra de la izquierda, escoge una de las plantillas que se muestran (como la de pulgadas para arquitectura o para carpintería).
2. Opcional: Si no ves la plantilla que quieres, selecciona Más plantillas en la parte superior derecha.

Aparecerán más opciones, como en la siguiente imagen. El texto en negrita describe para qué tipo de trabajo se ha creado este ajuste. Las unidades aparecen debajo del nombre de estilo. Cuando escojas una plantilla, aparecerá la ventana de modelado con esa plantilla aplicada.

Consejo: Puedes acceder a la ventana de bienvenida en cualquier momento mientras estás trabajando con SketchUp. Solo tienes que ir a la barra de menús y seleccionar **Ayuda > Bienvenida a SketchUp**. Cuando ya te hayas acostumbrado a crear modelos 3D en SketchUp, puedes crear plantillas personalizadas que se ajusten a tus preferencias.

2. Explorar la interfaz de SketchUp



Cuando se abre SketchUp, listo para que empieces a crear tu modelo 3D, verás una pantalla con lo siguiente:

- Barra de título
- Barra de menú
- Barra de Primeros pasos
- Zona de dibujo
- Barra de estado
- Cuadro de medidas
- Paneles por defecto

- **Barra de título**

La barra de título contiene los controles de ventana estándar (cerrar, minimizar y maximizar) y el nombre del archivo que tengas abierto. Cuando acabas de abrir SketchUp, el nombre de archivo abierto que se muestra es "Sin título", para indicar que aún no lo has guardado.

- **Barra de menú**

La mayoría de herramientas, comandos y configuraciones de SketchUp están disponibles en los menús de la barra de menús. Los menús son: SketchUp (solo en Mac), Archivo, Edición, Ver, Cámara, Dibujo, Herramientas, Ventana y Ayuda.

- **Barra de Primeros pasos**

Cuando empieces a usar SketchUp, verás por defecto la barra de Primeros pasos. Contiene las herramientas básicas que necesitas para empezar a crear modelos 3D.
Para mostrar otras barras de herramientas, selecciona **Ver > Barras de herramientas** . En la ventana de Barras de herramientas que se mostrará, selecciona las que quieras ver y pulsa Cerrar. En Mac OS puedes mostrar paletas de herramientas desde **Ver >**

Paletas de herramientas. Tienes más información sobre barras de herramientas y sobre cómo personalizarlas en la sección de Personalizar SketchUp del Centro de Ayuda.

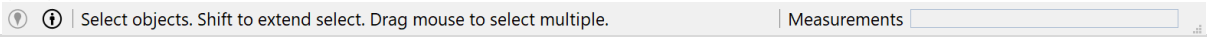
Consejo: Este artículo te presenta algunas herramientas básicas. Según vayas aprendiendo a crear modelos 3D en SketchUp, el instructor puede enseñarte (o recordarte) cómo utilizar cada herramienta. Consulta Aprender a usar las herramientas de SketchUp para más detalles.

- **Zona de dibujo**

El área de dibujo es donde creas el modelo. El espacio 3D del área de dibujo se identifica visualmente por los ejes de dibujo, que dan sensación de dirección tridimensional mientras trabajas.
El área de dibujo también puede contener un modelo sencillo de una persona para darte una sensación de espacio 3D.

- **Barra de estado**

Cuando empiezas con SketchUp, los dos elementos importantes en la barra de estado son las sugerencias de en medio y el cuadro de medidas a la derecha:
Consejo para usar las herramientas: En el centro de la barra de estado, haz clic en el icono de la interrogación para mostrar la ventana del instructor, que ofrece información básica sobre el uso de cualquier herramienta que selecciones en la barra de herramientas. El área central también tiene una descripción breve de la herramienta que tengas seleccionada. Esta área viene bien cuando no tienes claro cómo funciona una herramienta.
Cuadro de medidas: Este cuadro es una herramienta fundamental para crear modelos precisos. El cuadro muestra las dimensiones mientras dibujas. También lo puedes usar para modificar entidades seleccionadas (como crear una línea de una longitud concreta) o para crear copias de entidades a espacios regulares (como columnas, vallas... o los bloques de viviendas de una distopía postindustrial).



Consejo: ¿Te ha desaparecido el cuadro de medidas? Probablemente sea porque la ventana de SketchUp es más grande que el espacio que tienes en pantalla. Para volver a ver el cuadro de medidas, haz clic en el botón Maximizar de la barra de título.
Si estás usando Windows y tienes activada la ocultación de la barra de tareas, el cuadro de medidas se puede quedar por debajo cuando se muestra dicha barra. En este caso, verás el cuadro de medidas cuando termines con la barra de tareas y se vuelva a ocultar.
Consejo: En el lado izquierdo de la barra de estado encontrarás botones para geolocalizar y solicitar crédito. Estas opciones te ayudan a trabajar con las funciones avanzadas de SketchUp, que se escapan al ámbito de este artículo.

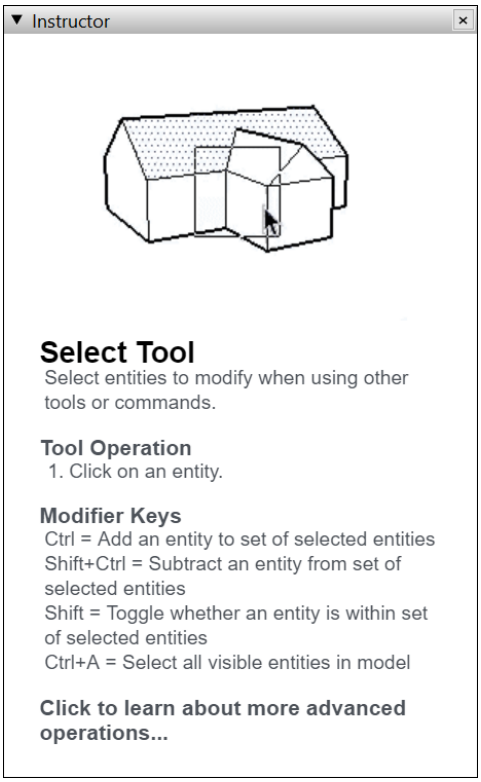
- **Paneles por defecto**

A la derecha de la pantalla verás un grupo de paneles que incluye Instructor, Materiales, Estilos y demás. El grupo por defecto aparece al abrir SketchUp, pero puedes cerrarlo haciendo clic en el botón de cerrar arriba a la derecha. Actívalo o desactívalo para que se muestre u oculte en el submenú Ventana > Bandeja por defecto.

3. Aprender a usar las herramientas de SketchUp

Mientras usas SketchUp, el instructor y la barra de estado te irán dando indicaciones sobre el uso de cada herramienta.
El instructor te enseña a usar la herramienta que tengas seleccionada. Para activarlo, como se muestra, selecciona **Ventana > Instructor**, que verás en la **bandeja por defecto**. Lo que ofrece el instructor:

- Una animación del uso básico de la herramienta seleccionada
- Una descripción de lo que hace la herramienta
- Pasos para usar la herramienta siguiendo la animación
- Teclas modificadoras que te permiten realizar más acciones
- Un enlace a artículos del centro de ayuda con funciones avanzadas de la herramienta



Si el instructor te da más detalles de los que te hacen falta, recuerda que también tienes consejos de uso en la barra de estado. Mira la sección de Barra de estado más arriba para más detalles.

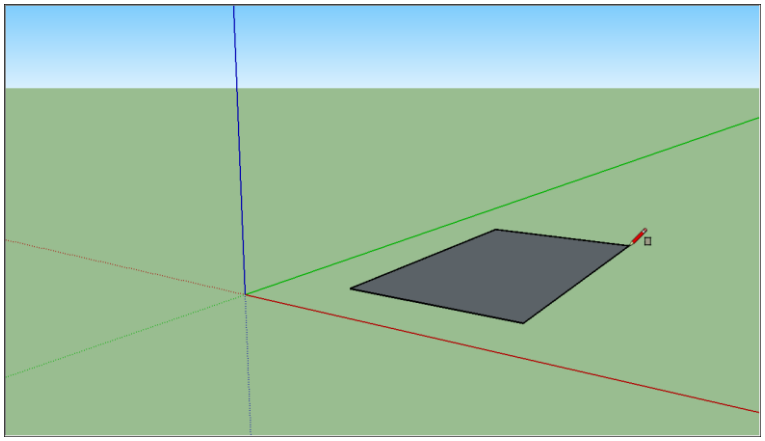
Ver la Quick Reference Card de SketchUp
La Quick Reference Card es una guía fácil de imprimir sobre todas las herramientas y teclas modificadoras de SketchUp. Tenla a mano cuando empieces a usar SketchUp y aprenderás a modelar de manera más rápida y eficaz. Este es el aspecto de la Quick Reference Card:
Para descargarte la Quick Reference Card en PDF, haz clic en el enlace de tu sistema operativo:

- Microsoft Windows
- macOS

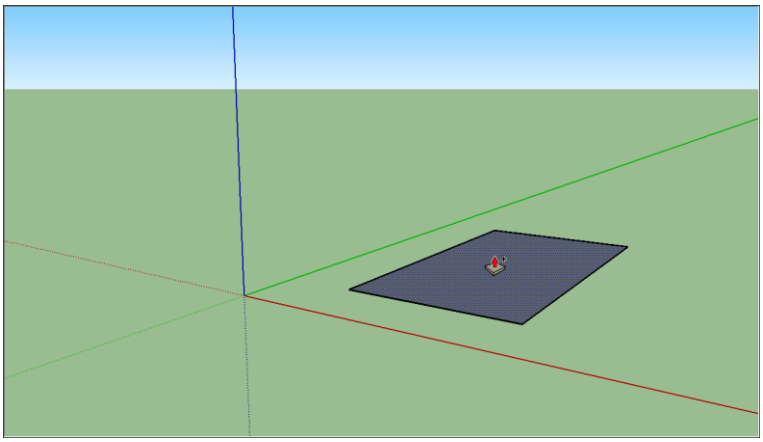
4. Crear tu primer modelo 3D en SketchUp

Si nunca has creado modelos en 3D en SketchUp (o en otro programa de modelado), estos pasos te dan una visión general de lo más básico:

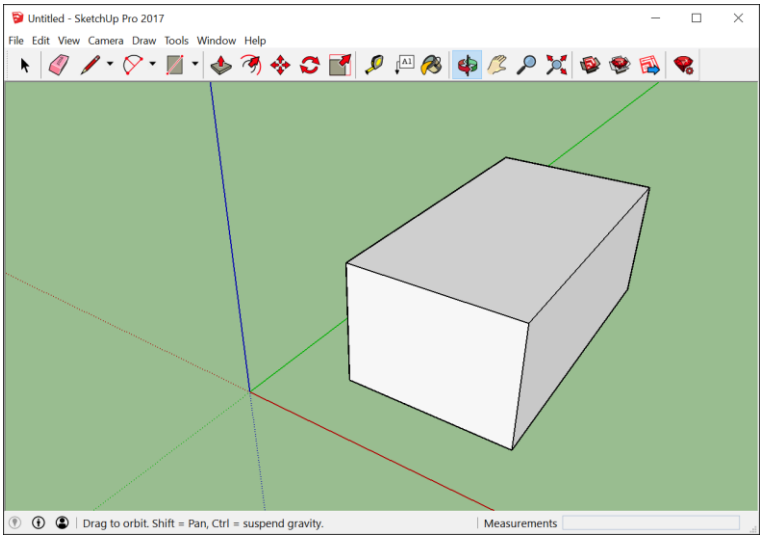
1. Selecciona la persona, haz clic derecho en la selección y selecciona **Eliminar** del menú contextual que aparece.
2. En la barra de herramientas de Primeros pasos, selecciona la herramienta **Rectángulo** ( ).
3. En el plano del suelo, entre los ejes rojo y verde, pulsa en el cursor de la herramienta **Rectángulo** ( ). Luego mueve el cursor a la derecha y haz clic otra vez. Aparecerá un rectángulo en el suelo como se muestra aquí.



4. En la barra de herramientas de Primeros pasos, selecciona la herramienta de **Empujar/tirar** ( ) y pon el cursor de Empujar/tirar sobre el rectángulo que acabas de crear, como se muestra en esta imagen.
5. Pulsa y arrastra tu rectángulo para crear una forma en 3D. No pierdas de vista el cuadro de medidas y suelta el cursor cuando la forma mida unos 5 pies de alto.



6. Sin hacer clic ni seleccionar nada, escribe **6'** y pulsa **Enter**. Verás que la altura de la forma cambia a 6 pies exactos y el valor que has escrito aparece en el cuadro de medidas.

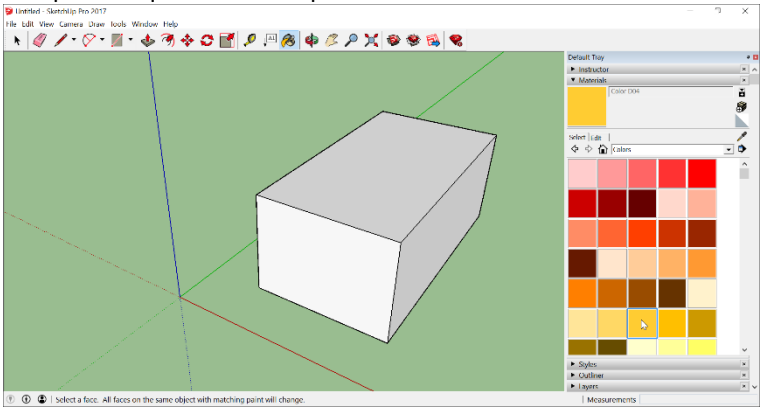


7. En la barra de herramientas de Primeros pasos, selecciona la herramienta **Órbita** ( ). Pon el cursor de Órbita sobre tu forma. Luego pulsa y mantén mientras bajas el ratón. Verás que la vista de tu forma cambia como se muestra en esta imagen. Prueba a pulsar y arrastrar con la herramienta Órbita tanto como quieras

8. En la barra de herramientas de Primeros pasos, selecciona la herramienta **Extensión de zoom** ( ). Si estás orbitando por ahí y pierdes de vista tu modelo, la herramienta de Extensión de zoom es una forma práctica de reorientarte.
9. Si tienes un ratón con rueda de desplazamiento, muévela hacia abajo para alejarte un poco. Trabajar en SketchUp es mucho más sencillo con un ratón con rueda de desplazamiento. Sin embargo, si tu ratón no tiene, pulsa en la herramienta de Zoom ( ) y podrás acercarte y alejarte así también.

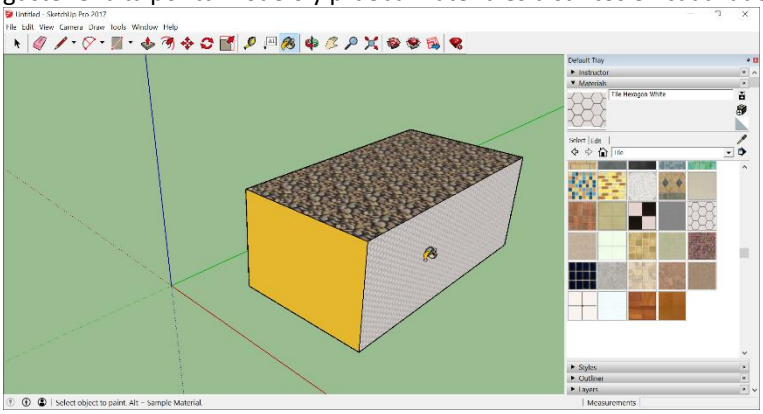
Consejo: Independientemente de la herramienta que tengas seleccionada, mantener pulsada la rueda de desplazamiento del ratón activa la herramienta Órbita hasta que la sueltes.

10. En la barra de herramientas de Primeros pasos, selecciona la herramienta Bote de pintura ( ).
11. En el panel de materiales que aparece, selecciona **Colores** del menú desplegable como se muestra. Luego elige un color de las opciones que salen en la pestaña Selección.

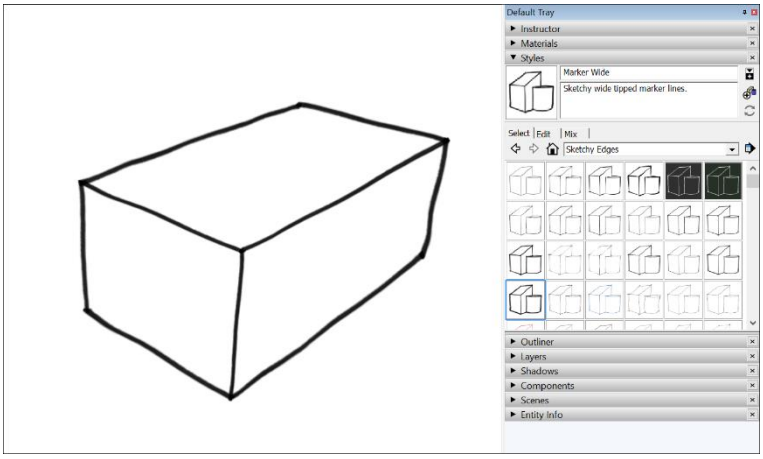


12. Pulsa en uno de los lados de tu modelo con el Cubo de pintura para aplicarle el color seleccionado. Haz un par de pruebas con distintas opciones del menú desplegable si te apetece. Por ejemplo, selecciona Paisajismo, Vallas y Vegetación del menú desplegable y ponle guijarros a tu modelo. Selecciona Mosaico del menú desplegable y aplica un patrón de mosaico que te

guste. Orbita por tu modelo y prueba materiales distintos en cada lado como se muestra.



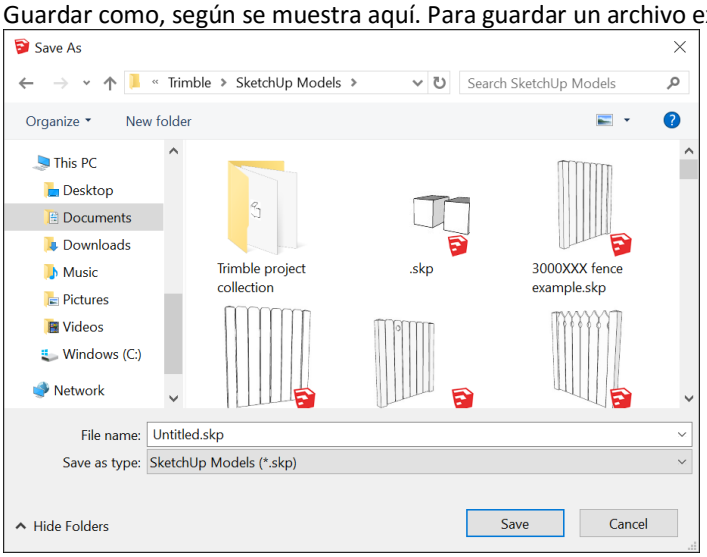
13. Cierra el panel de Materiales y selecciona **Ventana > Estilos**, que está en la **Bandeja por defecto**.
14. En el menú desplegable, selecciona **Bordes de boceto** y luego selecciona una opción de estilo. En la figura siguiente se ha seleccionado Rotulador ancho. Fíjate en que el estilo sobreescribe enteros los materiales y colores que hubiera aplicados. Para volverlos a ver, selecciona **En el modelo** del menú desplegable y luego pulsa en la opción de **Estilo simple**.



5. Guardar y reabrir modelos

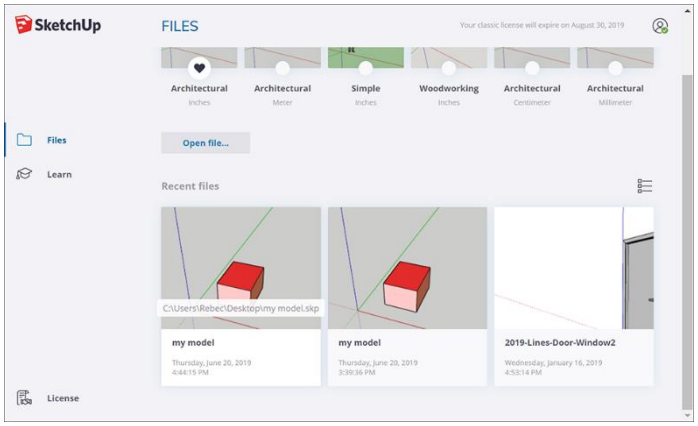
Para guardar tu modelo, sigue estos pasos:

1. En la barra de menús, selecciona **Archivo > Guardar**. Si es la primera vez que guardas un modelo, aparecerá la ventana de Guardar como, según se muestra aquí. Para guardar un archivo existente con otro nombre, selecciona **Archivo > Guardar como**.



2. Elige dónde quieres guardar tu modelo.
3. En el cuadro de Nombre de archivo, escribe un nombre para tu modelo. Los archivos de modelos de SketchUp usan .skp como extensión al final.
4. Opcional: Si quieres que tu modelo sea compatible con versiones anteriores de SketchUp, selecciona la versión en el menú desplegable de tipos de archivo de la ventana Guardar como.
5. Pulsa en el botón de **Guardar**.

Consejo: Cuando hayas guardado un modelo puedes volver a abrirlo en el futuro para seguir trabajando con él. Solo tienes que hacer doble clic en el archivo donde lo hayas guardado, o desde SketchUp seleccionar Archivo > Abrir. Si no tienes claro dónde se ha guardado un archivo, pasa el cursor por encima del archivo en la Ventana de bienvenida y podrás ver la ruta del mismo, como se muestra en esta imagen. O si has terminado tu modelo, puedes enseñarlo exportándolo como gráfico o creando una visita virtual.



6. Hacer una copia de seguridad de un archivo de SketchUp o restaurar un guardado automático

SketchUp guarda una copia de seguridad desde la *segunda* vez que guardes un archivo correctamente, y en cada guardado siguiente. Este archivo es una copia exacta de la versión del archivo guardada con anterioridad. La copia de seguridad se nombra según la regla NOMBREDEARCHIVO.skb en Windows y NOMBREDEARCHIVO~.skp en macOS, y se guarda en la misma carpeta que el archivo original.

Si SketchUp se cierra de forma inesperada mientras estás trabajando en un modelo, el archivo de recuperación no se borra. Por defecto, SketchUp guarda automáticamente tus archivos cada 5 minutos mientras estás trabajando activamente en ellos. Puedes recuperar el trabajo desde el momento del último guardado automático abriendo el archivo de recuperación. Para encontrar y abrir el archivo de recuperación, abre la ventana de Bienvenida de SketchUp, ve a la pestaña de Archivos y selecciona el archivo que quieras recuperar de la lista de Recientes.

SKETCHUP: dibujar el plano de una casa y construirla en 3D.

https://www.youtube.com/watch?v=n0DTMtLFaIY