



UNIDAD I

INTRODUCCIÓN A LA LÓGICA DE PROGRAMACIÓN

Algoritmos y su Representación

Competencias de Unidad:

- ▶ Definir el término algoritmo.
- ▶ Aplicar los conceptos y realizar práctica básicos relacionados con los algoritmos y su representación.
- ▶ Utilizar estructuras secuenciales y condicionales para analizar problemas.
- ▶ Diseñar algoritmos.

Expectativas de Logro:

- ▶ Identifican el tipo de problema a resolver de manera algorítmica.
- ▶ Representan los elementos básicos, diagramas de flujo de datos, pseudocódigos, utilizados para visualizar algoritmos en una computadora.
- ▶ Crean algoritmos.

Elementos de Competencia:

El estudiante es competente cuando:

- ▶ Comprende la utilización de algoritmos en la solución de problemas.
- ▶ Enlista los pasos que conlleva la solución de problemas de un algoritmo.
- ▶ Elabora algoritmos que se quieren ejecutar en una computadora mediante pseudocódigo y diagramas de flujo.

Contenidos:

- ▶ Algoritmos
- ▶ Pseudocódigos
- ▶ Diagramas de Flujo
- ▶ Programación Modular
- ▶ Programación Estructurada

Programación I

Introducción a la Lógica

Lógica Computacional

Es aquella aplicada a las ciencias de la computación que, con ayuda del raciocinio natural, existe la posibilidad de emitir incertidumbre en la verdad o falsedad de proposiciones, así mismo utiliza símbolos y métodos matemáticos para alcanzar análisis y optimización de los algoritmos.

Su uso es fundamental en los siguientes niveles:

- Circuitos Computacionales
- Programación Lógica
- Análisis y Optimización (de recursos temporales y espaciales) de Algoritmos

Todo sistema computacional, por muy complejo que sea, está compuesto por circuitos electrónicos que únicamente entienden un lenguaje binario (0 y 1); es allí donde la lógica computacional se encarga de modelar y optimizar tales sistemas a este nivel.

En la lógica computacional se dice que una variable tiene valor booleano cuando, en general, la variable contiene un 0 lógico o un 1 lógico. Esto, en la mayoría de los lenguajes de programación, se traduce en **false** (falso) o **true** (verdadero), respectivamente.

Tabla de Verdad (TV)

Es una tabla que muestra en forma sistemática los valores de verdad de una proposición compuesta en función de todas las combinaciones posibles de los valores de verdad de las proposiciones que la componen.

Cabe resaltar que una proposición es una oración que tiene un valor de verdad y existen dos tipos de proposiciones:

- **Proposiciones Simples:** Son aquellas que no se pueden dividir, **ejemplo: El árbol es verde.**
- **Proposiciones Compuestas:** Son aquellas que están formadas por dos o más proposiciones simples; es decir, si está afectada por negaciones o términos de enlace entre oraciones componentes, **ejemplo: Salí a correr pero me cansé.**

Dentro de un sistema de lógica proposicional, una interpretación no es más que una función que asigna un único valor verdadero a todas las fórmulas atómicas que están en consideración.

Una tautología es una fórmula bien formada que bajo cualquier interpretación de sus componentes atómicos tendrá valor de verdad igual a 1, es decir, verdadero. Por lo tanto, para determinar si una fórmula cualquiera es una tautología, basta con considerar todas las posibles interpretaciones de las fórmulas atómicas y calcular el valor de verdad del todo, esto se logra mediante una **Tabla de Verdad**.

En resumen, las tablas de verdad nos manifiestan los posibles valores de verdad de cualquier proposición, así como el análisis de las funciones de las proposiciones que la integran. A continuación, se explica de manera práctica cada uno de los símbolos utilizados en una **comparativa lógica**:

- **La Negación (\sim o \neg):** Es una proposición que tiene un **Valor de Verdad Opuesto** al original, es decir, "p" es verdadero, la negación será falsa, se denota como negación ($\sim p$) y se lee (no p) o (la negación de p).

Ejemplo:

- Romeo no conversa.
- Si p es la proposición, determine su negación:

$$p = 8 + 4 > 2 \quad V$$

$$\sim p = 8 + 4 < 2 \quad F$$

- $p =$ La multiplicación de un número por cero es diferente de cero F
- $\sim p =$ La multiplicación de un número por cero es igual a cero V

- **La Conjuntiva (\wedge):** Es una proposición compuesta que resulta de conectar dos proposiciones mediante la conjuntiva, esta proposición se denota por ($p \wedge q$) y se lee (p y q).

Ejemplo:

- Mariana cocinó pavo y Carlos le ayudó.
- ¿Cuál será el valor de verdad de: $6 > 3$ y $3 < 8$?

$$p = 6 > 3$$

$$q = 3 < 8$$

$$p \wedge q$$

$$V \wedge V$$

Respuesta: V

- 5 es un número negativo y natural
- $p =$ 5 es un número negativo
- $q =$ 5 es un número natural
- $p \wedge q$
- $F \wedge V$

Respuesta: F

- **La Disyuntiva Inclusiva (\vee):** Es una proposición compuesta que resulta de conectar dos proposiciones "p y q" mediante la disyuntiva inclusiva, se denota por ($p \vee q$) y se lee (p ó q).

Ejemplo:

- Te regalo agua o refresco. Te regalo azúcar y avena.
- Me traes naranjas y/o mangos.
- ¿Cuál será el valor de verdad de: $x^2 y^2 + x^3 y^2$ es x^2 ó $x^2 y$?

$$p = x^2$$

$$q = x^2y$$

$$p \vee q$$

$$V \vee V$$

Respuesta: V

- Q) $\sqrt{2}$ es racional ó 2 es impar

$$p = \sqrt{2} \text{ es racional}$$

$$q = 1 \text{ es impar}$$

$$p \vee q$$

$$F \vee V$$

Respuesta: V

- 6) **La Disyuntiva Exclusiva (O):** Es una proposición compuesta que resulta de conectar dos proposiciones "p y q" mediante la disyuntiva exclusiva, se denota por $(p \ O \ q)$ y se lee (O p ó q).

Ejemplo:

- Q) O me das naranjas o me das un mango.

- Q) ¿Cuál será el valor de verdad de: O $4 < 6$ ó $4 = 6$?

$$p = 4 < 6$$

$$q = 4 = 6$$

$$\begin{array}{cc} p & q \\ V & F \end{array}$$

Respuesta: V

- Q) O π es divisible por 2 ó π es múltiplo de 2

$$p = \pi \text{ es divisible por 2}$$

$$q = \pi \text{ es múltiplo de 2}$$

$$\begin{array}{cc} p & q \\ F & F \end{array}$$

Respuesta: F

- 6) **La Condicionante (\rightarrow):** Es una proposición compuesta que resulta de conectar dos proposiciones "p y q" mediante la condicionante, se denota por $(p \rightarrow q)$ y se lee (si p, entonces q).

Ejemplo:

- Q) Si te portas bien te doy una muñeca.

- Q) ¿Cuál será el valor de verdad de: Si $\frac{x}{4}$ es positivo, entonces $\frac{x}{4}$ es mayor que -1?

$p = \frac{3}{4}$ es positivo

$q = \frac{3}{4}$ es mayor que -1

$p \rightarrow q$

$V \rightarrow V$

Respuesta: V

Q Si $2^3 = 6$, entonces $3^2 = 6$

$p = 2^3 = 6$

$q = 3^2 = 6$

$p \rightarrow q$

$F \rightarrow F$

Respuesta: F

- 6 La Bicondicional (\leftrightarrow): Es una proposición compuesta que resulta de conectar dos proposiciones "p y q" mediante la bicondicional, se denota por $(p \leftrightarrow q)$ y se lee (p si y sólo si q).

Ejemplo:

Q Te llevo al supermercado si y sólo si me invitas a comer.

Q ¿Cuál será el valor de verdad de: $5 - 8 < 4$ si y sólo si 3 es un número primo?

$p = 5 - 8 < 4$

$q = 3$ es un número primo

$p \leftrightarrow q$

$V \leftrightarrow V$

Respuesta: V

Q 6 es un número impar si y sólo si 6 es un número primo

$p = 6$ es un número impar

$q = 6$ es un número primo

$p \leftrightarrow q$

$V \leftrightarrow F$

Respuesta: F

Analice el siguiente ejemplo utilizando el condicionante o proposición (v):

- O me da el chocolate o me da el refresco

p = Chocolate

q = Refresco

Explicación	p	q	$p \vee q$
1	V	F	V
2	F	V	V
3	V	V	V
4	F	F	F

Para explicar la tabla anterior, tomaremos el número de la columna (*Explicación*) para detallar el procedimiento efectuado para crear la tabla de verdad.

1. **Esta fila se interpreta:** $p = V$ significa que me da el chocolate; $q = F$ significa que no me da el refresco. Si consideramos la proposición (si me das un chocolate o un refresco te ayudo con las tareas) podríamos deducir que como nos dio el chocolate le ayudamos con las tareas aunque no nos haya dado el refresco, por ende $p \vee q = V$.
2. **Esta fila se interpreta:** $p = F$ significa que no me da el chocolate; $q = V$ significa que si me da el refresco. Si consideramos la proposición (si me das un chocolate o un refresco te ayudo con las tareas) podríamos deducir que como no me dio el chocolate pero si el refresco entonces, le ayudamos con las tareas, por ende $p \vee q = V$.
3. **Esta fila se interpreta:** $p = V$ significa que si me da el chocolate; $q = V$ significa que también me da el refresco. Si consideramos la proposición (si me das un chocolate o un refresco te ayudo con las tareas) podríamos deducir que como me dio el chocolate y también el refresco entonces, le ayudamos con las tareas, por ende $p \vee q = V$.
4. **Esta fila se interpreta:** $p = F$ significa que no me da el chocolate; $q = F$ significa que tampoco me da el refresco. Si consideramos la proposición (si me das un chocolate o un refresco te ayudo con las tareas) podríamos deducir que como no me dio el chocolate y tampoco el refresco, entonces no le ayudamos con las tareas, por ende $p \vee q = F$.

Nota:

- Si la tabla de verdad de la proposición es siempre verdadera, independientemente de la verdad o falsedad de las proposiciones simples, entonces la expresión es tautología.
- Si la tabla de verdad es siempre falsa, será una contradicción.
- Si es verdadera y falsa, la proposición es una contingencia.

Ejemplo de Tablas de Verdad

¿Bajo qué condiciones de p y q $[(p \leftrightarrow q) \wedge \neg q] \rightarrow (p \wedge \neg q)$ la proposición es falsa?

Solución:

Este ejercicio se resuelve con "tabla de verdad": En este caso, las proposiciones simples son dos, p y q . Luego, la tabla de verdad tiene $2^2 = 4$ filas.

- Empezamos llenando las columnas de las proposiciones simples, la tabla queda de la forma siguiente:

p	q	$\sim q$	$p \leftrightarrow q$	$(p \leftrightarrow q) \wedge \sim q$	$p \wedge \sim q$	$[(p \leftrightarrow q) \wedge \sim q] \rightarrow (p \wedge \sim q)$
V	V					
V	F					
F	V					
F	F					

- Luego, llenamos la tercera columna, que es la negación de la segunda y tenemos:

p	q	$\sim q$	$p \leftrightarrow q$	$(p \leftrightarrow q) \wedge \sim q$	$p \wedge \sim q$	$[(p \leftrightarrow q) \wedge \sim q] \rightarrow (p \wedge \sim q)$
V	V	F				
V	F	V				
F	V	F				
F	F	V				

- La cuarta columna, la llenamos usando el bi-condicionante para las columnas uno y dos, obteniendo:

p	q	$\sim q$	$p \leftrightarrow q$	$(p \leftrightarrow q) \wedge \sim q$	$p \wedge \sim q$	$[(p \leftrightarrow q) \wedge \sim q] \rightarrow (p \wedge \sim q)$
V	V	F	V			
V	F	V	F			
F	V	F	F			
F	F	V	V			

- Llenamos la quinta columna, usando la conjunción para las columnas cuatro y tres:

p	q	$\sim q$	$p \leftrightarrow q$	$(p \leftrightarrow q) \wedge \sim q$	$p \wedge \sim q$	$[(p \leftrightarrow q) \wedge \sim q] \rightarrow (p \wedge \sim q)$
V	V	F	V	F		
V	F	V	F	F		
F	V	F	F	F		
F	F	V	V	V		

- Para llenar la sexta columna, usamos la conjunción para las columnas uno y tres y tenemos:

p	q	$\sim q$	$p \leftrightarrow q$	$(p \leftrightarrow q) \wedge \sim q$	$p \wedge \sim q$	$[(p \leftrightarrow q) \wedge \sim q] \rightarrow (p \wedge \sim q)$
V	V	F	V	F	F	
V	F	V	F	F	V	
F	V	F	F	F	F	
F	F	V	V	V	F	

- Finalmente, para llenar la séptima y última columna de la tabla, usamos la condicionante para las columnas quinta y sexta, obteniendo:

p	q	$\sim q$	$p \leftrightarrow q$	$(p \leftrightarrow q) \wedge \sim q$	$p \wedge \sim q$	$[(p \leftrightarrow q) \wedge \sim q] \rightarrow (p \wedge \sim q)$
V	V	F	V	F	F	V
V	F	V	F	F	V	V
F	V	F	F	F	F	V
F	F	V	V	V	F	F

De esta forma, vemos que para que la proposición compuesta $[(p \leftrightarrow q) \wedge \neg q] \rightarrow (p \wedge \neg q)$ tenga valor de verdad “falso”, debe cumplirse que cada proposición simple, p y q, deben tener valor de verdad “falso”.

Tautologías

Es un vocablo griego que hace referencia a la repetición de un mismo pensamiento a través de distintas expresiones. Una tautología en la retórica es considerada una afirmación redundante. Las tautologías por lo general son consideradas como un error en el lenguaje o una falta de estilo; sin embargo, es posible apelar a las tautologías para enfatizar una cierta idea. **Por ejemplo:** “*Yo la vi haciendo la travesura con mis propios ojos*”, esta oración presenta una aclaración innecesaria acerca del uso de sus ojos, ya que no podría haber visto por otro medio.

De la misma manera podemos encontrar muchas oraciones comunes donde se presentan tautologías, como:

- Voy a subir arriba a buscar a María y le digo.
 - Tengo que salir afuera para hablar con Pedro.

En estas oraciones siempre que se sube es hacia arriba, de igual forma salir implica trasladarse hacia afuera de un lugar; por lo cual estas aclaraciones carecen de sentido y resultan innecesarias para la comprensión.

En el ámbito de la informática, las tautologías juegan un papel muy importante puesto que son una fórmula basada en un sistema de lógica proposicional que resulta verdadera para cualquier interpretación; es decir, que para cualquier asignación de valores será verdadera, para estos

casos se ve la necesidad de la construcción de una tabla de verdad y así determinar si una fórmula es una tautología o no.

Contingencias

Son proposiciones que pueden ser verdaderas o falsas, donde su veracidad o falsedad depende del conocimiento de los hechos o del momento cuando se dicen.

Por ejemplo: Hoy es Miércoles, esta afirmación sólo es cierta cuando es miércoles.

La veracidad de una proposición no solamente depende de la relación entre las palabras del lenguaje y los objetos en el mundo, sino también del estado del mundo y del conocimiento acerca de ese estado.

Ejemplo:

- Comprobar que $p \wedge (q \vee r)$.

p	q	r	$q \vee r$	$p \wedge (q \vee r)$
V	V	F	V	V
V	V	F	V	V
V	V	V	V	V
V	F	F	F	F
F	V	V	V	F
F	V	F	V	F
F	F	V	V	F
F	F	F	F	F

Contradicción

Es aquella proposición que en todos los casos posibles de su tabla de verdad, su valor siempre es falso (F). Su falsedad no depende de los valores de verdad de las proposiciones que la forman, sino de la forma en que están establecidas las relaciones sintácticas de unas con otras.

Ejemplo:

- Comprobar que la proposición $(p \wedge \neg p)$ es una contradicción.

p	$\neg p$	$p \wedge \neg p$
V	F	F
F	V	F

Estrategia de Aprendizaje # 1

Instrucciones: Realice en parejas, cada una de las actividades que se le muestran a continuación:

1. Elaborar la tabla de verdad de las siguientes proposiciones y decir en cada caso si se trata de una tautología, una contradicción o contingencia:
 - $\neg(p \Rightarrow \neg q) \Leftrightarrow (p \wedge q)$
 - $(p \wedge \neg q) \Leftrightarrow (\neg p \vee q)$
 - $p \Rightarrow (q \wedge r)$
 - $\neg p \Rightarrow (\neg p \wedge q)$
 - $(\neg p \wedge \neg q) \Rightarrow (p \vee q)$
 - $(p \Rightarrow q) \Rightarrow \neg p$
 - $(\neg q \wedge r) \vee (q \vee \neg r)$
 - $(r \wedge \neg r) \vee r$
2. Comprobar por medio de una tabla de verdad que las siguientes proposiciones compuestas son tautologías:
 - $p \vee \neg p$
 - $[p \wedge (p \Rightarrow q)] \Rightarrow q$
 - $(p \vee q) \Leftrightarrow (q \vee p)$
 - $(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$
 - $[(p \Rightarrow q) \wedge (q \Rightarrow r)] \Rightarrow (p \Rightarrow r)$

Introducción a la Programación

Cuando existe la necesidad de resolver de forma automatizada tareas que requieren de mucho análisis y múltiples procesos, por lo general existen aplicaciones o programas que son muy útiles para simplificar el trabajo; sin embargo, existen casos en que los programas de aplicación actuales no son suficientes para cubrir todas las necesidades. En el momento que se llega a este extremo, se hace necesaria la técnica de **Programación**, que consiste en el diseño paso a paso de una aplicación o grupo de programas que logren satisfacer todas las necesidades de una función o proceso determinado.

Programa: Es el conjunto de algoritmos capaces de ser entendidos por un ordenador y su función es la de permitir de manera alterna, la realización de tareas específicas sobre necesidades específicas. El software constituye toda la parte lógica total de un sistema informático.

Programación: Es el proceso mediante el cual se crea, diseña y codifica a través de un lenguaje de programación el código fuente de un software, hablando de manera más amplia, se puede decir que programar también es el arte de analizar, probar, depurar y en ocasiones hasta mantener el código fuente de un determinado software y aplicación que sea de total servicio para la satisfacción humana.

El objetivo central de la programación es la creación de aplicaciones, programas o software que sean de utilidad humana y generen la facilidad de realizar ciertas acciones cubriendo siempre una necesidad. La programación requiere conocimientos avanzados en áreas tales como:

- Lenguajes de programación
- Creación de algoritmos que generen la respuesta solicitada
- Análisis de aplicaciones ya diseñadas por otros programadores

En sus inicios los ordenadores sólo podían entender órdenes o instrucciones de un sólo lenguaje muy básico y específico basado en 1 y 0 conocido como **sistema binario**; para dar mayor uso a la programación, los primeros científicos programadores optaron por sustituir esa cadena de número simple por letras y luego palabras provenientes del idioma inglés, cuando lograron codificar ese conjunto de palabras establecieron un nivel de programación de mayor nivel denominado **Lenguaje Ensamblador**.

A medida que las tareas del ser humano se volvían complejas los programadores se vieron en la necesidad de crear nuevos métodos de programación denominados **Lenguajes de Programación de Alto Nivel**. De esta manera, lo que en el Lenguaje Ensamblador correspondía a un conjunto de instrucciones, en los Lenguajes de Alto Nivel significaban una sola instrucción y así la simplicidad de programar aumentaba. En la programación es muy importante que las instrucciones dictaminadas sean de manera clara dejando la ambigüedad a un segundo plano puesto que se maneja que, si un programa entre más claro es y menos ambiguo su funcionalidad sería más potente, bajo esta premisa podemos asegurar que cada algoritmo debe de ser totalmente legible para los lenguajes de programación.

Estrategia de Aprendizaje # 2

Instrucciones: Determine las diferencias entre los siguientes términos:



Programa	Programar	Programación

Algoritmo

Es un conjunto ordenado de instrucciones que deben de seguirse secuencialmente para la resolución de un problema o la satisfacción de una necesidad. Entendemos por algoritmo, cada paso que se sigue para resolver un problema dado. Los algoritmos son ejes centrales de la programación y según su claridad son llamados a dar vida al software o programa.

Características del Algoritmo

- **Preciso:** Indica el orden de realización en cada paso.
- **Definido:** Cuantas veces se ponga a prueba el algoritmo esa misma cantidad de veces debe mostrar el mismo resultado.
- **Finito:** El algoritmo debe de contener un número limitado de pasos.

Lenguajes Algorítmicos

Son una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso, los lenguajes algorítmicos se clasifican en:

- **Gráficos:** Se tratan de una presentación gráfica de las operaciones que realiza un algoritmo (*Diagramas*).
- **No Gráficos:** Es la representación descriptiva de las operaciones que realiza un algoritmo.

Ejemplos del Algoritmo

- **Necesidad u Problema:** Ver un partido de futbol en casa.
- **Desarrollo del Algoritmo:**
 1. Si el televisor no se encuentra encendido tomar el control de mando.
 2. Encender el televisor.
 3. Sintonizar el canal de transmisión del partido de futbol.
 4. Buscar un lugar adecuado para ver el partido.
 5. Disfrutar del partido de futbol.

El desarrollo algorítmico de este *ejemplo*, cumple con todos los requisitos descritos arriba, ya que cada paso precisa un orden con límite de pasos. Observando más detenidamente tenemos la condición **SI** en la primera instrucción en mayúscula que nos indica que si el televisor se encuentra encendido y la condición sería afirmativa, por lo tanto, tendríamos que omitir la instrucción 1 y 2.

Como ejercicio es recomendable algunos algoritmos de sucesos de la vida cotidiana para una mejor comprensión: encender el auto, ir al cine, etc.

Estrategia de Aprendizaje # 3

Instrucciones: En parejas, enumere ejemplos de algoritmos:

Ejemplos de Algoritmos



Solución de Problemas y Satisfacción de Necesidades en la Computadora

Aunque el surgimiento de una necesidad o de un problema sea inevitable, contamos con el proceso de diseñar programas según creatividad de los programadores; estos deben considerar una serie de fases o pasos comunes a seguir.

Al programar se insiste en que el algoritmo sea independiente del lenguaje de programación a utilizar. A continuación, se describe una serie de pasos a seguir para la realización de un programa agrupados en dos grandes grupos (*algoritmo e implementación*):

Algoritmo:

1. Definición del Problema
2. Análisis del Problema
3. Selección de la Mejor Alternativa
4. Diagramación
5. Prueba de Escritorio

Implementación:

6. Codificación
7. Transcripción
8. Compilación
9. Pruebas de Computadora
10. Documentación
11. Mantenimiento

Definición del Problema

Para resolver un problema, debemos formularlo con claridad. Si formulamos claramente el problema, tendremos más opciones para resolverlo.

Ejemplo: Imaginemos que se necesita un programa que despliegue un mensaje diciendo si el alumno está solvente o no con el pago de su colegiatura.

Para definir un problema debemos responder las siguientes preguntas:

- ¿Cuál es nuestro problema?
- ¿Dónde comienza?
- ¿Qué resultado deseamos obtener?

La definición del problema respecto al ejercicio anterior podría ser:

- Determinar si un alumno <>x</> está solvente o no.
- Mostrar en pantalla el estado de solvencia (solvente o en mora) del alumno.

Tomemos en cuenta que realizaremos programas para satisfacer necesidades y solucionar problemas de otras personas; por eso es importante conocer y entender lo que realmente otra persona vive, desea o necesita. En consecuencia, el enunciado del problema debe de ser claro y completo, ya que es importante conocer y entender exactamente lo que el usuario desea que haga la computadora; mientras esto no se comprenda no es recomendable continuar con la siguiente etapa.

Análisis del Problema

Es la etapa de estudio y preparación para la generación de soluciones que muestre la salida deseada. Para realizar el análisis debemos tomar en cuenta los siguientes puntos:

- Los datos o resultados esperados.
- Los datos de entrada necesarios para la obtención de los resultados.
- El proceso o método que se requiere para someter esos datos y lograr los resultados esperados, tomando en cuenta: áreas de trabajo, fórmulas y otros recursos importantes.

La buena definición y el buen análisis del problema presentan las bases para una buena programación y esta reside en entender la lógica del programa, es decir, determinar los pasos o fórmulas que el programa requiere y el orden en que deben realizarse.

Para el análisis de un programa es aconsejable responder las siguientes interrogantes:

- ¿Qué entradas tenemos?
- ¿Cuál es la salida deseada?

Si tomamos como ejemplo el ejercicio que hemos venido presentando, nos damos cuenta que hasta el momento tenemos como:

Definición del Problema:

- Determinar si un alumno <>x</> está solvente o no.
- Mostrar en pantalla el estado de solvencia (solvente o en mora) del alumno.

El Análisis del Problema sería:

- Dato de Entrada:

- ④ Nombre del alumno, mes, curso, sección, jornada, área.
- ④ PinAnual, mes.
- ⑥ Datos de Salida:
 - ④ Un mensaje que muestre el estado del alumno (alumno solvente o alumno en mora).

Selección de la Mejor Alternativa

Es el proceso durante el cual debemos seleccionar entre dos o más alternativas. Es muy probable que una vez analizado el problema tengamos muchas maneras de poder resolverlo, lo importante es lograr diferenciar cual es la mejor alternativa que determine una salida de datos esperada.

Si tomamos el ejemplo anterior nos damos cuenta que como método de entrada tenemos dos alternativas:

1. Nombre del alumno, mes, curso, sección, jornada, área.

Para conocer si esta es la mejor alternativa debemos evaluar los datos solicitados, **por ejemplo:** Podría ser que, al solicitar el nombre del alumno, nos encontramos con que hay más de dos personas que tienen ese mismo nombre. ¿Será esta la mejor solución?, no ya que si verificamos el pago de la persona que no corresponda entonces el resultado sería incorrecto.

2. PinAnual, mes

Si evaluamos los datos solicitados podríamos deducir que este sí nos devolverá los datos correctos ya que cada alumno tiene un PinAnual único que permite identificar el curso área, jornada, sección, nombre del alumno, etc.

Como método de salida tenemos solamente una alternativa, esta dependerá de la comparación del mes introducido a través de la entrada y verificación si el PagoMes está vacío o no; si está vacío o es igual a cero, significaría que no está solvente, de lo contrario estaría solvente.

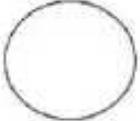
Diseño de Algoritmos (Diagramación)

Se refiere a la representación gráfica y lógica de la solución de un problema. Para la construcción de una casa lo primero que se hace es el diseño o sea el plano (instrucciones a seguir sobre el modelo y estilo) de la casa. Al igual que la construcción de una casa, para la elaboración de un programa, este debe ser diseñado previamente.

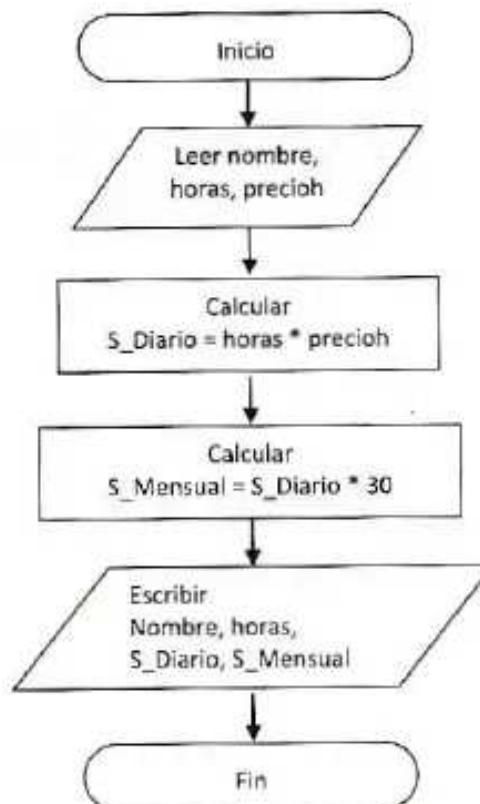
Diagramas de Flujo

Es la representación gráfica del flujo lógico de datos que se utilizará en la formulación de la solución de un problema, generalmente de una determinada parte del programa. Esto quiere decir que los diagramas se dibujan antes de escribir el programa para asegurar un desarrollo lógico.

Los símbolos más utilizados son los siguientes:

Función o Significado	Símbolo
1.- Proceso Proceso que lleva a cabo el ordenador; cualquier serie de transferencia de datos u operaciones aritméticas.	
2.- Entrada/Salida (E/S) Operación de entrada o salida, lectura y escritura de datos.	
3.- Decisión Verificar si el resultado de una expresión es verdadera o falsa.	
4.- Conector Punto de referencia que indica dónde debe continuar el diagrama de flujo. Se utiliza cuando deseamos indicar un cambio normal de datos (transferencia o bifurcación). También se utiliza como continuación en una nueva página.	
5.- Inicio/Fin Inicio y fin de un programa.	
6.- Salida de un Mensaje de Texto Presentación de resultados hacia una impresora o pantalla.	
7.- Sentido del Flujo de Datos Indica la siguiente operación a realizar y une un símbolo con otro.	

Ejemplo básico de un diagrama de flujo:



El diagrama anterior, representa la resolución de un programa que da como resultado el salario mensual de un empleado a partir de la lectura del nombre, horas trabajadas diariamente, precio por hora y el cálculo del sueldo diario.

Pseudocódigo

Es una herramienta utilizada para el diseño de programas permitiendo al programador expresar sus pensamientos en forma natural, mostrando el orden en que se ejecutarán las sentencias del programa sin ninguna ambigüedad.

En sí, es una mezcla de lenguaje de programación y de lenguaje natural. La idea del pseudocódigo consiste en aprovechar la flexibilidad y poder expresivo del lenguaje natural por un lado, así como las reglas de composición de los lenguajes de programación de alto nivel por el otro.

Reglas de los Pseudocódigos

- Utilizar palabras como Do While, EndDo o If, Else, EndIf, <nombre-proceso>, INVOCA <nombre-proceso>, End <Nombre-proceso>, las que tienen un significado especial, generalmente coinciden con las palabras correspondientes a los lenguajes de programación utilizados por los programadores para dar solución a un problema.
- Seguir reglas de sangrado para mostrar las dependencias dentro de cada uno de los segmentos que corresponden al diseño del programa.

El pseudocódigo generalmente consiste en un código falso (*pseudo*) o lenguaje natural, sin figuras. Las etapas de entrada, proceso y salida se manifiestan en términos que se utilizan en la comunicación.

PALABRA	UTILIZACIÓN	PALABRA	UTILIZACIÓN
ABRIR	Abre un archivo.	USUAL	Opcional en CASO.
INICIAR	Inicia un bloque de instrucciones.	NO	Niega la condición que sigue.
HAZ	Inicia bloque HAZ – HASTA.	O	Disyunción lógica.
ESCRIBIR	Visualiza un dato en pantalla.	CERRAR	Cierra un archivo.
ENTONCES	Complemento de SI – ENTONCES.	FIN	Finaliza un bloque.
LEER	Lee un dato del teclado.	HASTA	Cierra bloque HAZ – HASTA.

Al igual que un diagrama de flujo, un pseudocódigo debe indicar dónde comienza (**Inicio**) y dónde finaliza (**Fin**).

La resolución del ejercicio anterior sería:

Inicio

Leer nombre, horas, precioh

*S_Diario = horas * precioh*

*S_Mensual = S_Diario * 30*

Escribir Nombre, horas, S_Diario, S_Mensual

Fin

Prueba de Escritorio

Consiste en dar valores a las variables que hemos definido y que siguen el flujo del programa para comprobar si al final el resultado es el acertado.

Por ejemplo:

- No ingresar un dato de nacimiento que sea mayor que la fecha actual.
- No ingresar un número negativo donde deber ir uno positivo o sin decimales.
- No ingresar un valor numérico donde sólo debe ir texto.
- No ingresar un valor fuera del rango establecido.
- Etc.

En casos de generar algún error, el programa tendrá que enviar al usuario un mensaje indicándole esto y dándole la oportunidad de corregir.

En los casos en que deba ingresar una clave de acceso, no se debe permitir que la ingrese más de 3 veces. En este caso, se supone que no es la persona autorizada y el programa no debe permitirle seguir “**probando**” y normalmente lo cerramos sin más (luego de ponerle un mensaje en dónde le indicamos tal circunstancia).

Una vez que todo haya funcionado bien en el papel, entonces ya podemos escribir el pseudocódigo siguiendo el diagrama de flujo. Esta prueba es para no perder tiempo escribiendo el programa y luego tener que buscar en dónde está el error y como debemos corregirlo.

Si aplicamos valores a las variables del pseudocódigo o diagrama de flujo, podremos observar si el resultado es correcto o no.

Pseudocódigo:

Inicio

Leer nombre, horas, precioh
*S_Diario = horas * precioh*
*S_Mensual = S_Diario * 30*
Escribir Nombre, horas, S_Diario, S_Mensual

Fin

Aplicación de Valores:

Nombre = Sabdy Nazareth Aguilar
Horas = 8
Precioh = 50
*S_Diario = 8 * 50*
*S_Mensual = 400 * 30*

El resultado final sería:

<i>Nombre</i>	<i>horas</i>	<i>S_Diario</i>	<i>S_Mensual</i>
Sabdy Nazareth Aguilar	8	400	12000

Realice 3 o más pruebas para verificar que los resultados sean los correctos y si todo va bien el algoritmo sería el correcto.

Ejercicio Guiado

Planteamiento:			
Si 3 naranjas cuestan L. 9.00, ¿Cuántas podemos comprar con L. 99.00?			
Identificación del Problema:			
Entrada: Costo de 3 naranjas = L. 9.00 Cantidad de dinero con que se cuenta = L. 99.00		Salida: Total de naranjas que se pueden comprar con L. 99.00	
Planteamiento de Alternativas de Solución:			
Alternativa 1: $99 / 9 = 11$ $11 * 3 = 33$	Alternativa 2: $9 / 3 = 3$ $99 / 3 = 33$	Alternativa 3: $9 = 3$ $9 = 3$ $+9 = 3$ $99 = 33$	
Elección de una Alternativa:			
Alternativa 1: $99 / 9 = 11$ $11 * 3 = 33$	Seleccione la alternativa que considere más completa, sencilla o mejor, según las necesidades o el criterio de elección.		

Desarrollo de la Solución: (Algoritmo)	
1.- Inicio.	Entradas
2.- Obtener el precio de las 3 naranjas.	$3=L. 9.00$
3.- Obtener la cantidad de dinero con que se cuenta.	$L. 99.00$
4.- Dividir la cantidad de dinero entre el precio de las tres naranjas.	Proceso
5.- Multiplicar el resultado de la división por el número de naranjas que integran el precio.	$CN3 = 99 / 9$ $TN = 11 * 3$ $CN3 = \text{Cantidad de naranjas en bloques de tres}$ $TN = \text{Total de naranjas}$

6.- Mostrar el número de naranjas que se pueden comprar con la cantidad de dinero que se cuenta.	Salidas
7.- Fin	TN = 33
Evaluación de la Solución:	
99 / 9 = 11	Se comprueba que el resultado es correcto.
11 * 3 = 33	

Codificación

Consiste en colocar cada paso del diagrama previamente elaborado en una sentencia o instrucción en un lenguaje de programación, esto se realiza una vez hecha la prueba de escritorio y asegurarnos de que el proceso es válido, la codificación determina lo que en programación se denomina **Programa Fuente** (en inglés **Source**).

Todos los lenguajes de programación proporcionan facilidades para incluir líneas de comentarios o sugerencias con el objetivo de aclarar lo que se le ordena a la computadora y así entender mejor el programa, importante es saber que cada comentario no es ejecutado como instrucción.

Transcripción

Es el proceso mediante el cual convertimos un algoritmo en una secuencia de instrucciones legibles para la computadora.

Un algoritmo no puede ser ejecutado directamente, tiene que ser transcrita a lenguaje de máquina en forma de sentencia o instrucción para su ejecución. Estas instrucciones deben de estar sujetas a los parámetros de un lenguaje de programación.

En otras palabras, podemos decir que es el proceso a través del cual le escribimos al computador el programa que hemos acabado de escribir en papel. Para ello nos valemos de un programa llamado **Editor de Texto** que nos permite escribir un texto y grabarlo. Visto neutralmente, un programa no es más que un texto escrito bajo la óptica de algunas reglas preestablecidas por los creadores de un Lenguaje de Programación.

Compilación

Se refiere al proceso de traducción de un código fuente (escrito en un lenguaje de programación de alto nivel) a lenguaje máquina (código objeto) para que pueda ser ejecutado por la computadora.

La aplicación o herramienta utilizada para la traducción del código fuente a lenguaje máquina se llama **compilador**. El compilador o traductor realiza un análisis interno del código fuente para detectar errores de sintaxis ocasionados por fallas en la codificación o transcripción.

Cualquier falla de lógica que pueda contener nuestro programa fuente no será detectada por nuestro compilador. Cuando no hay errores graves de compilación, cada instrucción del programa fuente se traduce a lenguaje de máquina creando el **Programa Objeto**.

Algunas computadoras utilizan interpretadores, (generalmente para el Lenguaje Basic) en reemplazo de programas compiladores. La diferencia radica en que el interpretador recibe desde una terminal sólo una instrucción a la vez, la analiza y si está correcta la traduce al formato propio de máquina, en el caso de que la instrucción tuviera un error el interpretador llama la atención del programador para que corrija la instrucción.

La compilación ofrece varios resultados, los cuales enlistamos a continuación:

- Listado del programa fuente
- Listado de los errores detectados
- Listado de campos utilizados

Los errores se deben corregir sobre el mismo programa fuente. La compilación se debe repetir hasta eliminar por completo los errores y obtener un programa óptimo que pueda ser ejecutable.

Pruebas de Computadora

Son las pruebas que se realizan una vez finalizado el programa (ya contamos con el código fuente) antes de ser entregado al usuario final o cliente.

Una vez que tenemos el programa ejecutable (lenguaje máquina), ejecutamos el programa e introducimos datos de prueba como se hizo en la prueba de escritorio, al analizar los resultados puede ocurrir cualquiera de las siguientes situaciones:

- La lógica del programa está bien, pero hay errores sencillos, estos se pueden corregir modificando algunas instrucciones o incluyendo unas nuevas; en este punto, debemos repetir el proceso desde el paso 8.
- Hay errores ocasionados por fallas en la lógica, lo que nos obliga a regresar a los pasos 5 y 6 para una revisión exhaustiva y modificación del diagrama.
- Hay errores muy graves y lo más aconsejable es que regresemos al paso 2 para analizar nuevamente el problema y repetir el proceso.
- No hay errores y los resultados son los esperados.

Con las pruebas de computadora nos damos cuenta de que el funcionamiento del programa terminado es el correcto, y así eliminamos todo tipo de errores.

Ejemplo:

Si se creó un programa para la suma de dos números ingresados por el usuario, pero al introducir las órdenes a la computadora se escribió:

Obtener el primer número (n1)

Obtener el segundo número (n2)

Calcular la suma = n1 * n2

Escribir suma

Al realizar las pruebas con los datos de prueba ($n1 = 2$ y $n2 = 2$) y verificar el resultado no detectamos ningún error ya que la respuesta de la suma y de la multiplicación en este caso siempre será 4, pero si $n1 = 3$ y $n2 = 3$ observamos que el resultado a mostrar es 9 y que, por lo tanto, no tiene lógica puesto que el resultado de la suma sería 6.

Con este *ejemplo*, nos damos cuenta de la importancia de realizar pruebas de computadora con el producto ya terminado. Es de suma importancia realizar pruebas con distintos tipos de datos, incluyendo situaciones poco comunes pero que pueden presentarse en la ejecución del programa; de esa manera tendremos más probabilidades de erradicar los errores en el programa.

Documentación

Es toda la información ya sea en papel o incluida directamente en el código del programa, desde el proceso de definición del problema hasta su ejecución y su posterior mantenimiento.

La documentación puede ser interna y externa. La **documentación interna** es la contenida en líneas de comentarios dentro del código, esta se escribe a la hora de programar; la **documentación externa** está compuesta por los análisis, estudios de factibilidad, diagramas de flujo, pseudocódigos, manuales de usuario y cualquier tipo de información que se incluya en el proceso de programación que no esté escrito dentro del código.

La documentación es vital cuando se desea corregir posibles errores (futuros) o bien hacer cambios de cualquier índole en el programa. Cualquier cambio que se realiza una vez instalado y ejecutado el programa debe de ser documentado, a esto se le llama **mantenimiento del programa**. Cuando se tiene listo el programa para ser ejecutado es necesario que hagamos su documentación externa siguiendo ciertos estándares y normas de la instalación así como también recomendaciones.

Una buena documentación para el buen funcionamiento del programa deberá incluir:

- Enunciado del problema o necesidad
- Narrativo con la descripción de la solución

- Relación de las variables utilizadas por el programa junto con su función
- Diagrama del programa
- Lista de la última compilación
- Resultados de la ejecución
- Recomendaciones de mantenimiento

Mantenimiento

Son todas las actividades relacionadas con el programa que se llevan a cabo después de terminado el mismo, estas actividades garantizan una estabilidad en su funcionamiento y mejoras de ser posible. Consiste en correcciones y en algunos casos en modificaciones para que el programa no interrumpa sus funciones para las cuales está hecho. Este proceso puede durar días, meses y hasta años.

Estrategia de Aprendizaje # 4

Instrucciones: Liste y explique con sus propias palabras los pasos para la solución de problemas:



Objetivos de la Programación

Al programar se deben de tener objetivos claros de lo que necesitamos obtener y de cómo lo vamos a obtener, creando soluciones que signifiquen una alternativa en todos los puntos del programa, lo importante no es comenzar con un programa muy complejo sino comenzar con un programa pequeño pero eficiente.

A continuación se muestran los objetivos principales que se deben tomar en cuenta al momento de programar:

1. Exactitud
2. Claridad
3. Eficiencia

Exactitud: Un programa debe de satisfacer su función principal y específica de manera exacta. Es muy común que la complejidad de la labor final del programa puede influir directamente en la satisfacción de algunas funciones específicas. El programa como materia terminada debe de ser igual de efectivo que cada una de sus funciones individuales.

Claridad: Un programa debe de ser en su totalidad claro, complejo únicamente lo necesario. La claridad del programa es una ayuda importante para el programador mismo en el diseño y limpieza del programa y hacia otros que tengan que leer y alternar el programa en alguna etapa posterior.

La claridad de un programa se determina de igual manera que se obtiene la claridad de cualquier documento escrito y para lograrlo se requiere:

- a. Una separación lógica del texto en partes compresibles (módulos, estructuras, capítulos, secciones) que reflejen la distinción entre los temas que se describen.
- b. Una selección cuidadosa de las características del lenguaje usadas en cada parte para expresar su sentido propuesto tan preciso como sea posible.
- c. Selección cuidadosa de las palabras usadas para denotar los objetos y conceptos involucrados.
- d. Inclusión de comentarios y preámbulos para clarificar el texto principal cuando sea necesario.
- e. Un aprovechamiento de los dispositivos para presentación de textos, tales como líneas en blanco, sangría etc. Todo esto con el fin de enfatizar en aspectos y relaciones importantes.

Eficiencia: Es la capacidad del programa de mostrar los resultados de manera rápida y sin la utilización de muchos recursos en la computadora, cabe señalar que la eficiencia se mide de 2 formas:

- a. Por el tiempo que la computadora usa para poder leer y mostrar el resultado esperado según secuencia de instrucciones.

- b. Por la cantidad de memoria de la computadora usada en mostrar ese resultado. Tomemos en cuenta que en ocasiones nuestro programa competirá con otros por el uso de los recursos de la computadora.

Elementos utilizados en la Solución de un Programa

Datos

Son los diferentes objetos de información con los que un programa trabaja.

Tipos de Datos

Es un atributo de una parte de los datos que indica al ordenador sobre la clase de datos que va a procesar; esto incluye poner restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

Todos los datos, tienen un tipo asociado con ellos que se utiliza para conocer con qué información trabajará; es decir, cuando se ingresa el precio de un producto, es necesario que este contenga decimales, al solicitar la edad de una persona esta debe introducirse con números enteros, etc.

La asignación de tipos a los datos tiene tres objetivos principales:

- Detectar errores de operaciones en los programas.
- Determinar cómo ejecutar las operaciones.
- Utilizar de manera indicada las herramientas programables.

Los tipos de datos son importantes porque nos indican el uso de los datos correctos, para el caso un recipiente cualquiera sería de mucha ayuda a la hora de guardar nuestros implementos personales en nuestra habitación, pero lo adecuado es utilizar una cómoda para guardar nuestros objetos puesto que fue diseñada para eso.

Clasificación de Datos

Los más comunes más utilizados en los lenguajes de programación son:

- **Numéricos:** Dentro de estos se puede hacer mención de los tipos enteros, reales o coma flotante y los exponentiales.

Ejemplo:

1, 9884657738, 11.5, 1.5 x 215

- **Alfanuméricos (Carácter):** Dentro de este tipo tenemos (a, A, C, &, etc.) que responden al código ASCII (American Standard Code Information Interchange, Código Estándar Norteamericano para

el intercambio de información), y son conocidos como cadenas de caracteres o strings, siempre son escritos entre comillas.

Ejemplo:

"Bienvenidos a Ediciones Fares", "15 De Septiembre Del 2021"

- **Lógicos:** Los tipos lógicos solamente toman los valores verdadero o falso.

Estrategia de Aprendizaje # 5

InSTRUCCIONES: Investigue, la función principal de los datos con los que un programa trabaja:

	Función Principal de los Datos	 EDICIONES Fares Una mirada hacia el futuro

Identificadores, Variables y Constantes

Identificadores

Es un conjunto de caracteres que se utilizan para identificar cualquier entidad del programa, los identificadores pueden ser combinaciones entre letras y números. Los identificadores representan los nombres de los objetos de un programa (constantes, variables, tipos de datos, procedimientos, funciones).

Los identificadores deben seguir las siguientes reglas:

- Deben comenzar con letra o "_" y no pueden contener espacios en blanco.
- Letras, dígitos y caracteres subrayados ("_") están permitidos después del primer carácter.
- No se deben utilizar caracteres especiales como %, \$, #, /, etc.
- Se recomienda que sea corto, de preferencia menos a 255 caracteres.

Como conclusión, podemos decir que un identificador es un método que se utiliza para nombrar las celdas de memoria en la computadora en lugar de la dirección de memoria.

Constantes

Son todos los valores que no están sujetos a cambios en la comprobación del algoritmo ni en la ejecución del programa, reciben un valor y este no puede ser modificado.

Ejemplo:

Se desea asignar un identificador para almacenar el valor del número π:

$\text{Pi} = 3.141592$

Pi es el identificador y se le asigna el valor constante 3.141592 a través del signo igual.

Variables

Son los espacios de memoria que se pueden modificar durante la ejecución del programa. Contrario de las constantes, estas reciben un valor que puede ser modificado cuantas veces sea necesario.

Ejemplo: Nota1, Promedio, sueldo, precio

Al momento de nombrar variables es necesario tomar en cuenta estas reglas básicas:

- Los nombres de variable puede ser una palabra que contenga guiones u otro carácter, con excepción de espacios.

Ejemplo:

Se pretende seleccionar un identificador para una variable que almacenará las horas trabajadas de un empleado en la semana, se puede expresar de la siguiente manera:

Horas_Trabajadas

H_Trabajadas

HorasTrabajadas

HT

- Los nombres de las variables deben de tener significados adecuados. En el inciso anterior, usamos HT para nombrar Horas Trabajadas, resulta muy práctico, pero no tan conveniente. Un nombre adecuado ayuda a seguir la lógica de nuestro programa.

Ejemplo:

TOTAL = INVERSION * TIPO_INTERES (CORRECTO)

REAL MADRID = HONDURAS15SEPTIEMBRE * FRANCISCO MORAZÁN (INCORRECTO)

Vemos que en el segundo caso nos encontramos totalmente con asignaciones de nombres incongruentes que al final no representan la lógica de un programa.

- c. Los nombres de las variables no deben de comenzar con un dígito.

Ejemplo:

Podemos comenzar con Hora1 pero nunca 1Hora.

Sentencias de Asignación

Se utilizan para asignar o almacenar valores en variables o constantes.

Es una operación que sitúa un valor determinado en una posición de memoria. Se demuestra en pseudocódigo con el símbolo “=”. Su estructura Identificador = expresión.

El tipo de expresión debe ser del mismo que el de la variable o constante, en caso contrario, durante la fase de compilación se producirá un error de tipos. Aunque a la fecha existen lenguajes de programación con una potente herramienta para la conversión de datos como: Visual Basic, no es recomendable asignar valores de un tipo diferente a variables que son de otro.

Reglas de Asignación:

- Una variable en el lado derecho de una sentencia de asignación debe tener un valor antes de que la sentencia de asignación se ejecute. Si x no tiene un valor antes de ejecutar $p=x+1$ o $p:=x+1$ se producirá un error lógico.
- A la izquierda de una sentencia de asignación solamente pueden existir variables.

Ejemplo: No es válido escribir $3625.00 = \text{precio}$, se debió escribir $\text{precio} = 3625.00$

Hay que recordar que la operación de asignación es destructiva debido a que el valor almacenado en una variable se pierde o se destruye sustituyéndose por el nuevo valor en la sentencia de asignación.

Operadores utilizados en Programación

En todos los lenguajes de programación se utilizan operadores para efectuar operaciones aritméticas. Combinando las variables y constantes en expresiones aritméticas por medio de funciones adecuadas. Una expresión es un conjunto de datos o funciones unidos por operadores aritméticos.

Operadores Aritméticos

Son los utilizados para realizar operaciones aritméticas básicas:

Símbolo	Significado	Símbolo	Significado
+	Suma	/	División
-	Resta	^	Potenciación
*	Multiplicación	Mod o %	Residuo de la división

Reglas de Evaluación y Realización de las Operaciones

- Todas las subexpresiones entre paréntesis se evalúan primero. Las subexpresiones entre paréntesis anidados se evalúan de adentro hacia afuera, es decir, que el paréntesis más interno se evalúa primero.
- *Prioridad de Operaciones:* Dentro de una misma expresión o subexpresión, los operadores se evalúan en el siguiente orden:

Símbolo	Operación
* , /	Multiplicación y división
div, mod	División y módulo de enteros
+, -	Suma y resta

- Los operadores en una misma expresión o subexpresión con igual nivel de prioridad se evalúan de izquierda a derecha.

Operadores Relacionales

Son utilizados para la comparación de dos o más valores; para realizar comparaciones entre distintos valores se usan los siguientes operadores relationales:

Símbolo	Significado	Símbolo	Significado
<	Menor que	>=	Mayor o igual que
>	Mayor que	=	Igual a
<=	Menor o igual que	<>	Distinto a, diferente de

Nota: El símbolo igual (=) se puede utilizar para asignación o comparación dependiendo el lugar donde se encuentre dentro de la expresión.

Toda expresión comparativa tendrá un valor lógico.

Ejemplo:

<i>Si x = 50</i>	<i>Si y = 25</i>	<i>Si z = 3</i>
$x < y$	es FALSO	
$x = y$	es FALSO	
$z = 3$	es VERDADERO	
$y + z = x$	es FALSO	
$x + y = 75$	es VERDADERO	

Operadores Lógicos

Las expresiones lógicas pueden combinarse para formar expresiones más complejas utilizando los operadores lógicos and, or y not.

<i>Operador</i>	<i>Significado</i>
And	y, Conjunción
Or	O, Disyunción
Not	No, Negación

Ejemplo:

Si mañana, el día amanece con sol **Y** me queda tiempo iré a correr, si el día **NO** amanece con sol, me iré al cine, pero si llueve **O** hace frío me quedaré en casa.

Los operadores lógicos se utilizan con constantes lógicas.

[operando1] **operador** [operando2]

El operando1 según el tipo de operador puede no existir.

La operación **and** combina dos condiciones simples y produce un resultado verdadero sólo si los dos operandos son *verdaderos*. La operación **or** es verdadera si uno de los dos operandos es *verdadero*. La operación **not** actúa sobre una sola condición simple u operando y simplemente niega (o invierte) su valor. Cuando la expresión lógica contiene varios tipos de operadores, es preciso seguir un nuevo orden de precedencia o prioridad para obtener el valor final de la expresión.

<i>Prioridad</i>	<i>Operador</i>
1	(exp), not
2	*, /, div, mod, and
3	+, -, or
4	<, <=, =, <>, >=

Operador And:

<i>Exp 1</i>	<i>Exp 2</i>	<i>Resultado</i>
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

Operador Or:

<i>Exp 1</i>	<i>Exp 2</i>	<i>Resultado</i>
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Operador Not:

<i>Exp 1</i>	<i>Resultado</i>
Verdadero	Falso
Falso	Verdadero

Ejemplo:

(exp1 < exp2) && (exp2 < exp3)

El símbolo “**&&**” representa la conjunción lógica **And**.

Es verdadero, si ambas son verdaderas, pero si una o ambas son falsas el resultado es falso.

(exp1 < exp2) | | (exp2 < exp3)

El símbolo “**| |**” representa la conjunción lógica **Or**.

Es verdadera, si una es verdadera, pero si ambas son falsas el resultado es falso.

!(exp1 < exp2)

El símbolo “**!**” representa la negación **Not**.

Es falso, siempre que sea verdadero que *(exp1 < exp2)*; y es verdadera si la comparación es falsa, **Not** actuando sobre *(exp1 < exp2)* es equivalente a: *(exp1 >= exp2)*.

!(exp1 == exp2)

Da como resultado verdadero, si *exp1* es distinto de *exp2*, y es falsa si *exp1* es igual a *exp2*, esto es equivalente a: *(exp1 != exp2)*

Operaciones de Entrada y Salida de Datos

Son las operaciones que permiten el proceso de leer determinados valores y asignarlos a determinadas variables, almacenarlos y luego mostrar en pantalla los resultados.

Los datos se pueden almacenar en memoria de tres formas diferentes:

1. Asociados con constantes
2. Asignados a una variable, con sentencia de asignación
3. Una sentencia de lectura

La tercera forma de almacenamiento es la más indicada cuando se manipulan diferentes datos al ser ejecutado el programa. La lectura de datos permite asignar valores desde dispositivos hasta archivos externos en memoria, a este se le conoce como **Operación de Entrada o Lectura**.

La operación de entrada/lectura se representa así:

- Pseudocódigo
Leer (Lista de Variables)
- Diagrama de Flujo

Lista de
Variables

En la medida que se realizan cálculos en el programa, es necesario visualizar los resultados; a esto se le conoce como **Operación de Escritura o Salida**. La operación de salida / escritura se representa:

- Pseudocódigo
Escribir (Lista de Variables, "Mensaje")
- Diagrama de Flujo

Lista de Variables
"Mensaje"

En una instrucción de salida pueden incluirse mensajes de texto y variables.

Ejemplos:

1. Crear un algoritmo que solicite al usuario ingresar los catetos de un triángulo rectángulo y muestre en pantalla la hipotenusa.

Análisis de Necesidades:

- Identificar el inicio.
- Enviar un mensaje solicitando el ingreso de la medida de los catetos.
- Lectura del dato de entrada.
- Almacenamiento del número en la memoria, utilizando x, y, z como variables.
- Proceso a realizar $c = \sqrt{a^2 + b^2}$.
- Salida en pantalla mostrando la medida de la hipotenusa.
- Fin.

<i>Diagrama de Flujo</i>	<i>Pseudocódigo</i>
<pre> graph TD Inicio([Inicio]) --> IngresarPrimerCateto["Ingresar primer cateto:""] IngresarPrimerCateto --> a[/a/] a --> IngresarSegundoCateto["Ingresar segundo cateto:""] IngresarSegundoCateto --> b[/b/] b --> Proceso["c = sqrt(a^2 + b^2)"] Proceso --> Salida["La hipotenusa es: c"] Salida --> Fin([Fin]) </pre>	Inicio ESCRIBIR "Ingresar primer cateto;" Leer a ESCRIBIR "Ingresar segundo cateto;" Leer b HACER $c = \sqrt{a^2 + b^2}$ ESCRIBIR "La hipotenusa es;" c Fin

2. Crearemos un algoritmo que sume dos números.

<i>Diagrama de Flujo</i>	<i>Pseudocódigo</i>
<pre> graph TD Inicio([Inicio]) --> Inicializar["a=0, b=0, c=0"] Inicializar --> IngresarPrimerNumero["Ingresar primer número:""] IngresarPrimerNumero --> a[/a/] a --> IngresarSegundoNumero["Ingresar segundo número:""] IngresarSegundoNumero --> b[/b/] b --> Suma["c = a + b"] Suma --> Salida["La suma es: c"] Salida --> Fin([Fin]) </pre>	Inicio ESCRIBIR "Ingresar primer número" Leer a ESCRIBIR "Ingresar segundo número" Leer b HACER $c = a + b$ Escribir "La suma es;" c Fin

Estrategia de Aprendizaje # 6

Instrucciones: Según lo antes visto, crear los siguientes algoritmos con su pseudocódigo y su respectivo diagrama de flujo:



Creación de Algoritmos



- Un algoritmo que multiplique dos dígitos introducidos por el usuario.
- Un programa para determinar si una persona ante la ley de Honduras es, o no, mayor de edad.
- Crear un algoritmo que divida dos números.
- Crear un algoritmo que realice la siguiente fórmula: $2x^2 + 3x^3 - 5$
- Un algoritmo que muestre la multiplicación de tres números.

Errores al Ejecutar un Programa

Es importante mencionar, que ningún programador está exento de errores y que hasta los programadores más experimentados los cometen. Al ejecutar un programa se pueden presentar 3 tipos de errores:

1. **Errores de Compilación:** Se producen por el uso incorrecto de las reglas del lenguaje de programación, casi siempre por errores de sintaxis.
2. **Errores en Tiempo de Ejecución:** Se producen por instrucciones que la computadora puede comprender pero no ejecutar. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error.
3. **Errores Lógicos:** Se producen en la lógica del programa y la fuente del error es el diseño del algoritmo, son más difíciles de detectar puesto que el programa puede funcionar y no producir errores de compilación ni ejecución, pero mostrará resultados incorrectos.

Estilos de Programación

Se entiende por estilos de programación los métodos que existen para manejar la calidad de los programas de computación.

Programación Estructurada

Esta es una técnica utilizada para organizar programas dentro de una colección de pequeños módulos aislados. Los módulos pueden ser agrupados en una jerarquía o en una red y cada unidad tiene una entrada única y un punto de salida. El procesamiento de líneas de código se ejecuta de arriba hacia abajo.

La programación estructurada nos permite construir un producto de software en grupo, en otras palabras, varios programadores pueden trabajar en diferentes partes en la elaboración de un producto.

La programación estructurada se basa en los siguientes puntos:

1. Diseño del programa de lo general a lo particular.
2. Todo programa puede ser diseñado utilizando únicamente las tres estructuras básicas:
 - Secuencial
 - Alternativa
 - Repetitiva

Esto ayuda a que los programas sean fácilmente leídos por cualquier persona, que el número de errores en el proceso de programación sea bajo, que los programas sean autodocumentados, y que puedan mantenerse fácilmente.

La programación estructurada hace los programas más fáciles de escribir, verificar, leer y mantener; utiliza un número limitado de estructuras de control que minimiza la complejidad de los problemas.

Ventajas de la Programación Estructurada

Elaborar programas de computadora sigue siendo una labor que demanda esfuerzo, creatividad, habilidad y cuidado. Con este estilo podemos obtener las siguientes ventajas:

- Los programas son más fáciles de entender, ya que un programa bien estructurado puede ser leído en secuencia sin necesidad de saltar de un sitio a otro en la lógica.
- Reducción del esfuerzo en las pruebas, el programa se puede tener listo para producción normal en un tiempo menor al tradicional.
- Reducción de los costos de mantenimiento.
- Programas más sencillos y más rápidos.
- Aumento en la producción del programador.

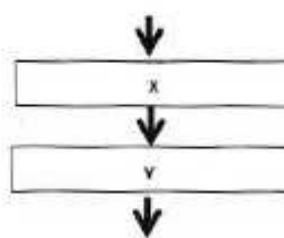
- Se facilita la utilización de las otras técnicas para el mejoramiento de la productividad en programación.
- Los programas quedan mejor documentados internamente.

Estructuras de Control

Las estructuras de control de un lenguaje de programación se refieren al orden en que las instrucciones de un algoritmo se ejecutarán. Un programa estructurado se compone de funciones, segmentos, módulos y sub rutinas, cada cual con una sola entrada y una salida. A cada módulo se denomina “**Programa Apropiado**”. Como mencionamos antes, la programación estructurada tiene un teorema central, el cual afirma que cualquier programa, no importa el tipo de trabajo que ejecute puede ser elaborado utilizando únicamente las 3 estructuras básicas de control lógico. Las estructuras básicas de control lógico son:

1. **Secuencial:** Son las estructuras que indican que las instrucciones de un programa se ejecutan una después de la otra en el mismo orden en el cual aparecen en el programa.

Se representa gráficamente por símbolos de flujo continuo en ambos con una sola entrada y una sola salida.



En este ejemplo la instrucción Y depende de la instrucción X, si el orden fuese diferente esto daría un error lógico.

Ejemplo:

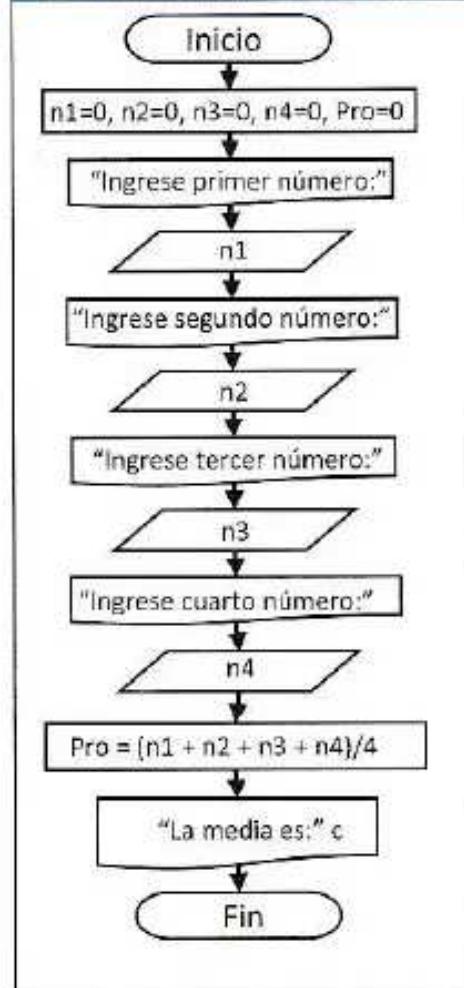
Crear un algoritmo que determine la media de 4 números.

Definición del Problema:

- Necesitamos la media o promedio de 4 números que el usuario introduzca.

Análisis del Problema:

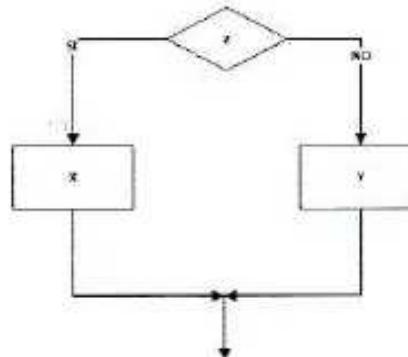
- Se tendrá como entrada cuatro números (n_1, n_2, n_3, n_4).
- Como salida el promedio de los cuatro números; $Pro = (n_1, n_2, n_3, n_4) / 4$.

Diagrama de Flujo	Pseudocódigo
 <pre> graph TD Inicio([Inicio]) --> Inicializar[n1=0, n2=0, n3=0, n4=0, Pro=0] Inicializar --> Preg1["Ingresar primer número."] Preg1 --> Entrada1[/n1/] Entrada1 --> Preg2["Ingresar segundo número."] Preg2 --> Entrada2[/n2/] Entrada2 --> Preg3["Ingresar tercer número."] Preg3 --> Entrada3[/n3/] Entrada3 --> Preg4["Ingresar cuarto número."] Preg4 --> Entrada4[/n4/] Entrada4 --> Calcular[Pro = (n1 + n2 + n3 + n4) / 4] Calcular --> Resultado["La media es: " & Pro] Resultado --> Fin([Fin]) </pre>	<p>Inicio</p> <p>ESCRIBIR "Ingrese primer número:"</p> <p>LEER n1</p> <p>ESCRIBIR "Ingrese segundo número:"</p> <p>LEER n2</p> <p>ESCRIBIR "Ingrese tercer número:"</p> <p>LEER n3</p> <p>Escribir "Ingrese cuarto número:"</p> <p>LEER n4</p> <p>HACER media = (n1 + n2 + n3 + n4) / 4</p> <p>ESCRIBIR "La media es:" Pro</p> <p>Fin</p>

2. **Selección (decisión):** Controlan la ejecución de otras instrucciones en los algoritmos. Las ejecuciones de ciertas instrucciones pueden efectuarse mediante la evaluación de los datos que representa una condición, tal como la siguiente: una computadora puede estar encendida, la velocidad de un automóvil puede incrementarse hasta un punto en que no se pueda detener.

Las estructuras de decisión permiten tanto ejecutar la instrucción como desviar o evitar su ejecución.

También son conocidas como **si/sino (if/else)** o **si-cierto-falso**, que plantean la selección entre dos alternativas con base en el resultado de la evaluación de una condición o predicado. Equivale a la instrucción **If** de todos los lenguajes de programación. Se representa gráficamente de la siguiente manera:



En el diagrama anterior Z representa una condición o estructura lógica (Verdadero / Falso), si Z se evalúa como verdadero, se ejecuta la instrucción representada por X; pero si se evalúa como falsa, se ejecuta la instrucción representada por Y.

La estructura del diagrama tiene una sola entrada y una sola salida, las funciones X, Y representan cualquier estructura básica o conjunto de estructuras.

El pseudocódigo y la sentencia de decisión queda de la forma siguiente:

SI condición lógica ENTONCES

Sentencia X

SINO

Sentencia Y

Ejemplos:

- Crear un algoritmo para determinar si un alumno está reprobado o aprobado, ingresando nota de la clase y tomando que se aprueba con 70%.

Definición de Problema:

- El problema consiste en tomar la decisión del mensaje que se mostrará, para lo cual se evaluará la nota ingresada, si esta es mayor o igual a 70 se mostrará el mensaje “Aprobado”, de lo contrario se mostrará el mensaje “Reprobado”.

Análisis del Problema:

- Se tendrá como entrada la nota del alumno.
- Como salida el mensaje de Aprobado o Reprobado; este se mostrará dependiendo de la comparación de la nota con el 70%.

Diagrama de Flujo	Pseudocódigo
<pre> graph TD Inicio([Inicio]) --> Nota0[Nota = 0] Nota0 --> Ingresar["Ingrese la Nota:"] Ingresar --> Nota[/Nota/] Nota --> Decision{Nota >= 70} Decision -- V --> Aprobado["Aprobado"] Decision -- F --> Reprobado["Reprobado"] Aprobado --> Fin([Fin]) Reprobado --> Fin </pre>	Inicio ESCRIBIR "Ingrese la Nota;" LEER Nota SI Nota >= 70 ENTONCES ESCRIBIR "Aprobado" SINO ESCRIBIR "Reprobado" Fin

- b) Crear un algoritmo con su diagrama de flujo y pseudocódigo del siguiente problema: El estado de Honduras a través del Hospital Escuela reparte a diario 2200 vacunas contra el tétano y necesita un programa que muestre un mensaje de alerta cuando se tiene un mínimo de 100 vacunas en existencia.

Definición del Problema:

- El problema consiste en crear un programa que tire una alerta cuando se reporte 100 vacunas en existencia.

Análisis del Problema:

- Como entrada tenemos 2200 vacunas con las que comienza a diario el Hospital Escuela.
- Como salida tendremos un mensaje de alerta de que sólo hay en existencia 100 vacunas; para la realización de este ejercicio se toma en cuenta que se descargan de una en una las vacunas del inventario.

Diagrama de Flujo	Pseudocódigo
<pre> graph TD Inicio([Inicio]) --> Vacunas0[Vacunas = 0] Vacunas0 --> VacunasA[Vacunas a Retirar] VacunasA --> Vacunas[/Vacunas/] Vacunas --> Existencia[Existencia = Existencia - Vacunas] Existencia --> Cond{Existencia > 100} Cond -- F --> MensajeF[Sólo hay 100 vacunas en existencia] MensajeF --> Fin([Fin]) Cond -- V --> MensajeV[Hay vacunas suficientes en existencia] MensajeV --> Fin </pre>	<p>Inicio</p> <p>ESCRIBIR "Vacunas a Retirar"</p> <p>LEER Vacunas</p> <p>HACER Existencia - Vacunas</p> <p>SI existencias >= 100</p> <p>ESCRIBIR "Hay vacunas suficientes en existencia"</p> <p>SINO</p> <p>ESCRIBIR "Sólo hay 100 vacunas en existencia"</p> <p>Fin</p>

If Aninado: A una sentencia If se le llama aninada cuando la sentencia de la rama verdadera o la sentencia de la rama falsa es a su vez una sentencia If.

Las sentencias If aninadas se utilizan para implementar en un programa decisiones con más de dos alternativas.

Ejemplo:

La obesidad es un problema que nos puede afectar a todos, el peso está estrechamente relacionado con la altura y peso. Se creó una fórmula que se le denomina Índice de Masa Corporal (IMC), este se obtiene a partir del peso y altura de una persona. Para conocer el estado de obesidad de una persona se deben tomar en cuenta los siguientes datos:

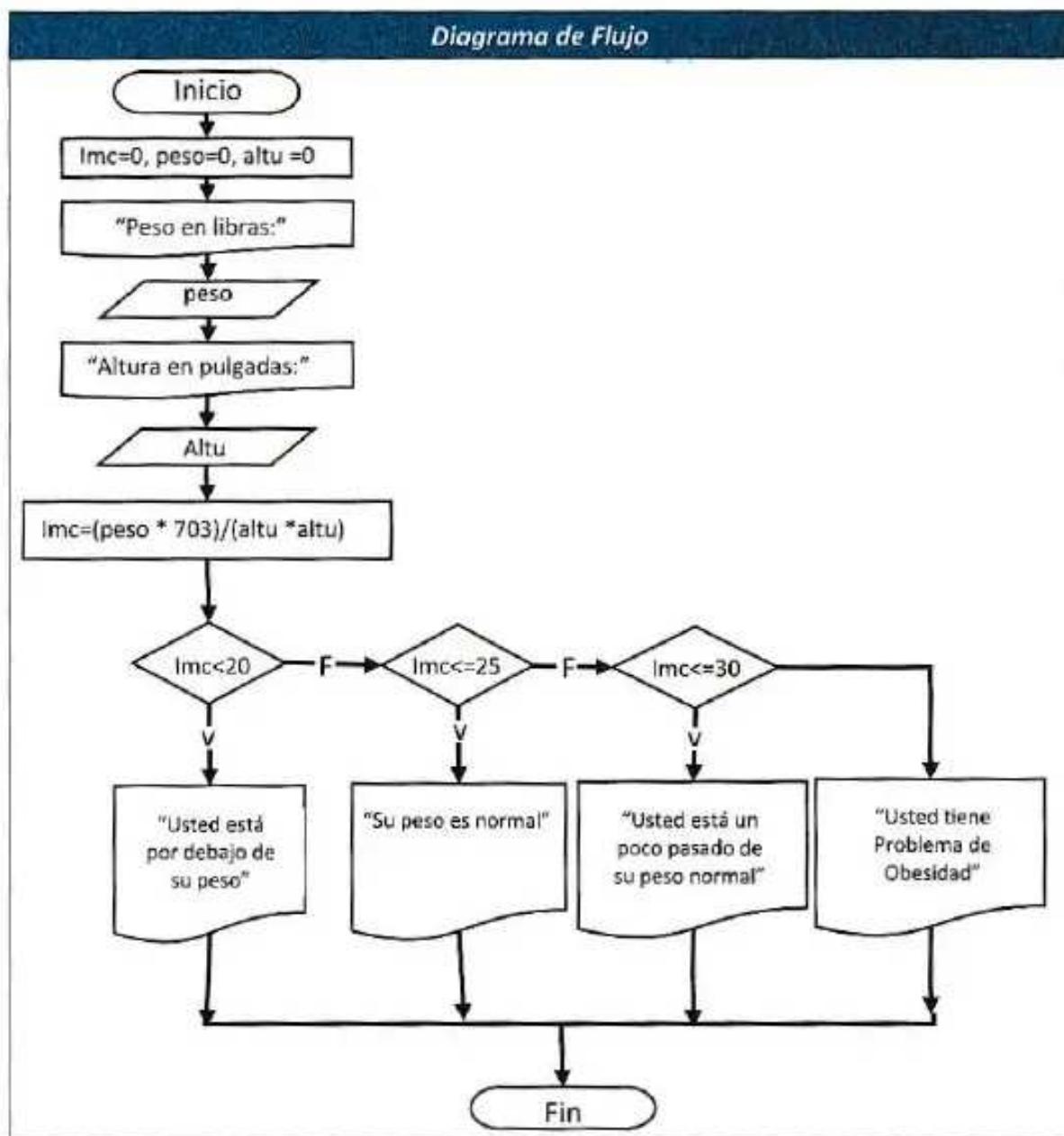
- Si el IMC es menor que 20, está debajo del peso normal.
- Si el IMC está entre 20 y 25, es normal.
- Si el IMC está entre 26 y 30, un poco pasado de su peso normal.
- Si el IMC es mayor a 30 tiene obesidad.

Definición del Problema:

- El problema consiste en obtener los datos de altura y peso de la persona, y a partir del IMC determinar el grado de obesidad.

Análisis del Problema:

- Tendremos como entrada el peso y lo representaremos con peso, la altura que la representaremos con Altu.
- Como salida uno de los mensajes (Está debajo del peso normal, es normal, un poco pasado de su peso normal, tiene obesidad).
- Lo haremos calculando primero el IMC representado por IMC con la fórmula $IMC = (\text{Peso} \times 703) \div (\text{altura}^2)$ y luego comparándolo con los valores de interpretación del enunciado para decidir qué mensaje mostrar.

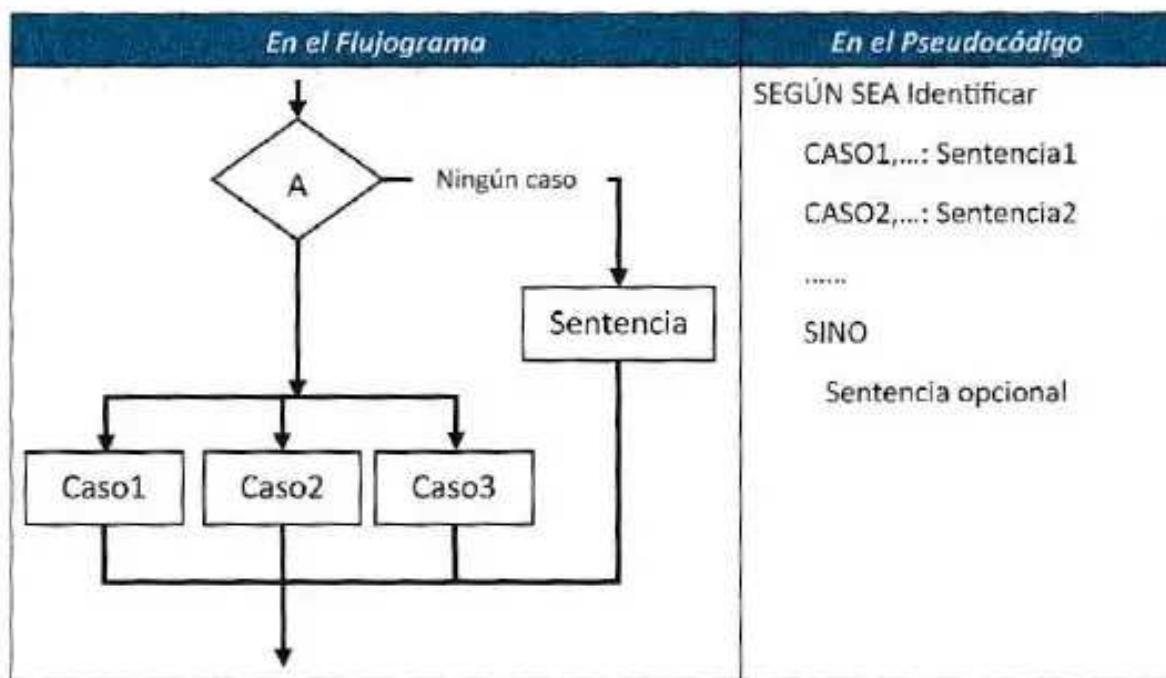


Pseudocódigo
Inicio
ESCRIBIR "Peso en Libras"
LEER peso
ESCRIBIR "Altura en Pulgadas"
LEER altu
HACER Imc = (peso * 703) / (altu * altu)
SI Imc < 20 ENTONCES
ESCRIBIR "Usted está por debajo de su peso"
Sino
Si Imc <= 25 ENTONCES
ESCRIBIR "Su peso es normal"
Sino
Si Imc <= 30 ENTONCES
ESCRIBIR "Usted está un poco pasado de su peso normal"
Sino
ESCRIBIR "Usted tiene Problema de Obesidad"
Fin

Sentencia de Selección Según Sea (case): Esta sentencia se utiliza para elegir entre diferentes alternativas. Esta se compone de varias sentencias simples, pero cuando se ejecuta, sólo una de ellas se selecciona y ejecuta.

El valor de selector debe ser un tipo ordinal, y los valores constantes deben tener el mismo tipo que el selector.

La sentencia de selección según sea, se utiliza en vez de los If anidados.

**Ejemplo:**

En la biblioteca de un Centro de Educación se cobra por libro según el área, 300 lempiras libro de español, 350 libro de matemáticas, 400 libro de informática y 470 libro de laboratorio de informática.

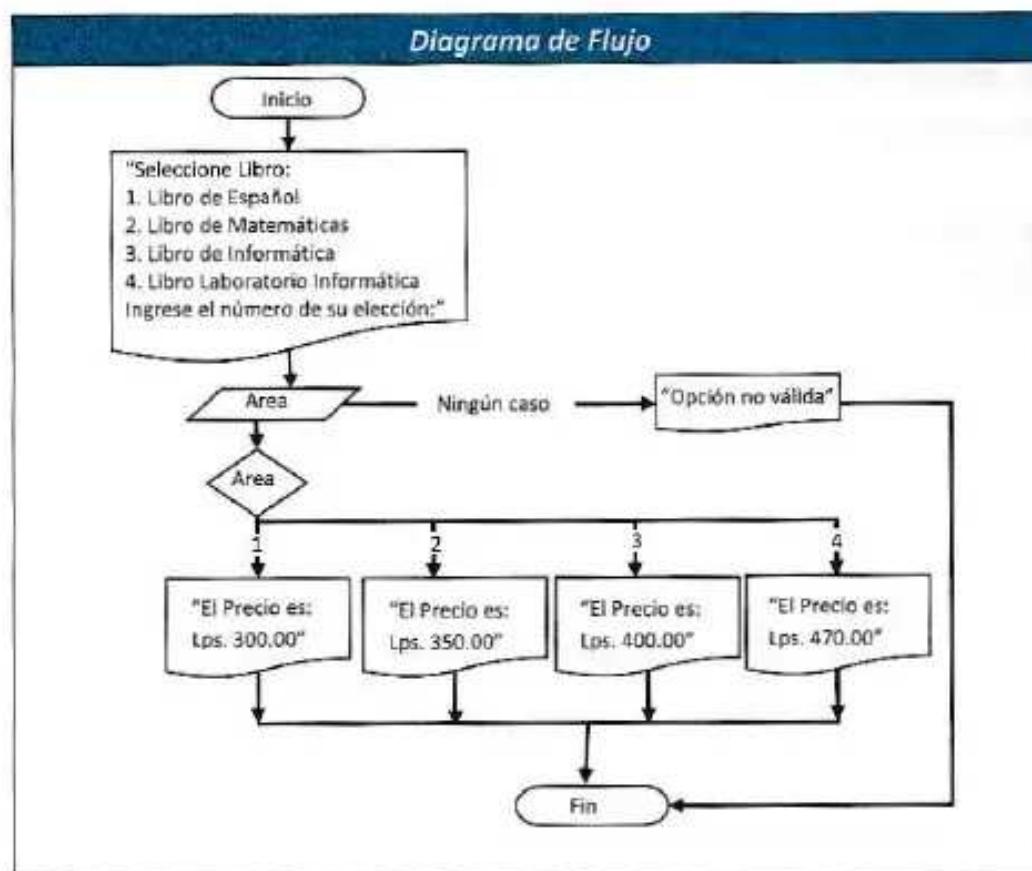
Crear un algoritmo que muestre el precio del libro a comprar por el alumno.

Definición del Problema:

- El problema consiste en ingresar el área del libro y que el algoritmo determine el precio del libro.

Análisis del Problema:

- Tendremos como entrada el área del libro representada como Área.
- Como salida tenemos el precio.
- El proceso se utilizará una estructura según sea para determinar el valor.



Pseudocódigo

```

Inicio
  ESCRIBIR "Ingrese el número de su elección:"
  LEER Area
  SEGÚN SEA Area
    CASO = "1"
      ESCRIBIR "El precio es: Lps. 300.00"
    CASO = "2"
      ESCRIBIR "El precio es: Lps. 350.00"
    CASO = "3"
      ESCRIBIR "El precio es: Lps. 400.00"
    CASO = "4"
      ESCRIBIR "El precio es: Lps. 470.00"
    SINO
      ESCRIBIR "Opción no valida"
Fin
  
```

- 3. Estructura Repetitiva (Iteración):** Es una estructura de control que permite la repetición de una serie determinada de sentencias, se denomina bucle (lazo o ciclo). El cuerpo del bucle contiene las

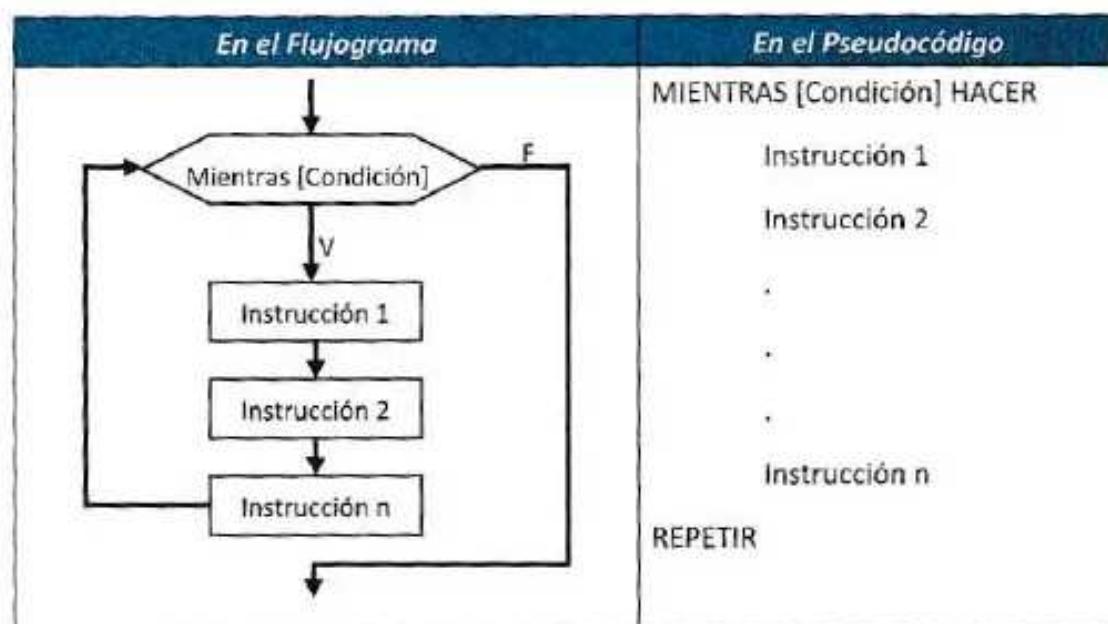
sentencias que se repiten. Las computadoras están especialmente diseñadas para ejecutar tareas repetitivas, las estructuras de control repetitivas son aquellas en las que una sentencia o grupos de sentencias se repiten muchas veces.

El conjunto de instrucciones que se repiten en un ciclo se denominan **cuerpo del ciclo** y cada repetición del cuerpo del ciclo se denomina **iteración**.

A continuación definiremos algunos conceptos que intervienen en una iteración:

- **Contador:** Es una variable que tiene como trabajo contar las iteraciones, es de tipo numérico el cual aumenta o disminuye su valor de manera manual o automática.
- **Acumulador:** Es una variable que almacena los valores lógicos intermedios.
- **Inicialización:** Es la asignación de un valor de inicio al contador y acumulador.

Mientras – Hacer (While): Permite que un grupo de instrucciones se pueda ejecutar varias veces mientras se cumple una determinada condición. Mientras una condición se mantenga verdadera su función es repetir un bloque de instrucciones.



Ejemplo:

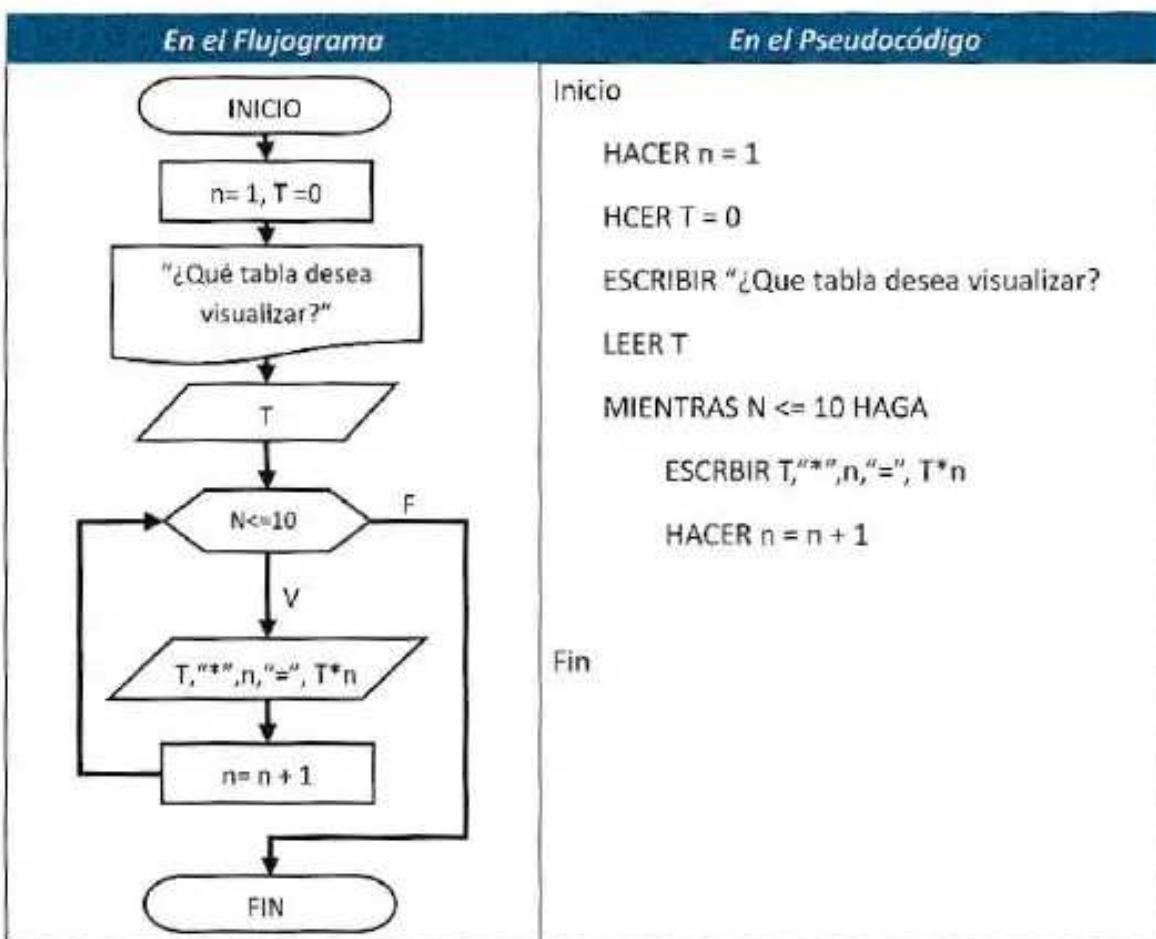
Crear un algoritmo que muestre la tabla de multiplicar según el número introducido por el usuario, la tabla se realizará del 1 hasta el 10.

Definición del Problema:

- El problema consiste en mostrar la tabla de multiplicar que el usuario desea.

Análisis del Problema:

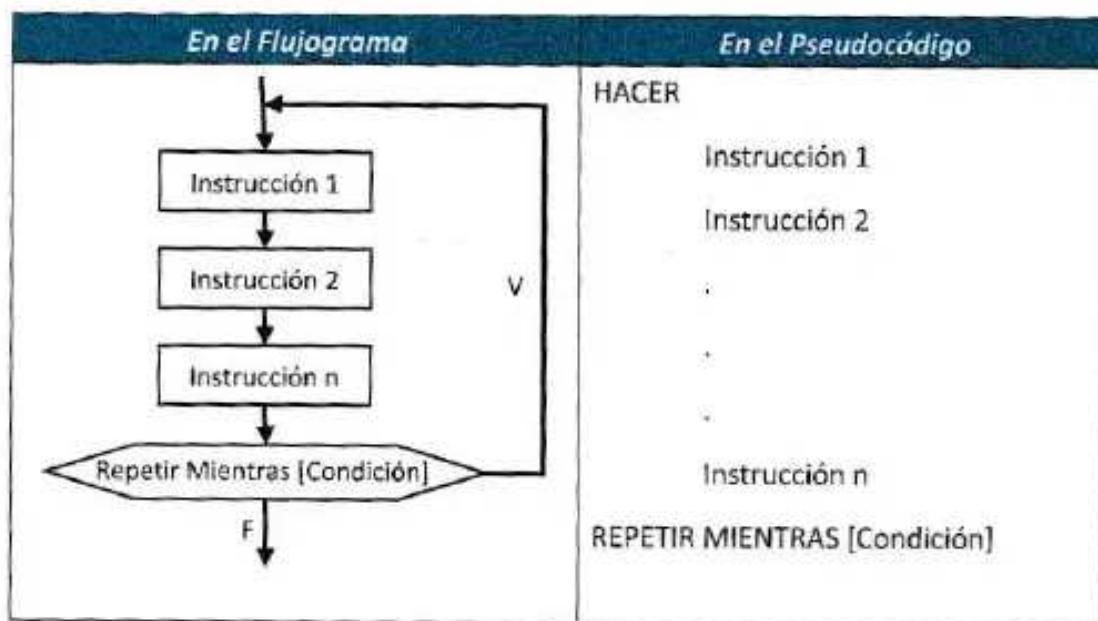
- Tenemos como dato de entrada el número introducido por el usuario.
- Como salida tenemos la tabla de multiplicar.



Hacer – Repetir Mientras (Do While): Es una estructura que garantiza la ejecución de un bloque de sentencias como mínimo una vez, en esta estructura el número de iteraciones no se conoce por anticipado y el cuerpo del bucle se repite mientras se cumple una determinada condición, por esta razón a estos bucles se les conoce como **bucles condicionales**. Ahí radica su diferencia con Mientras... Hacer, ya que esta puede llegar a no ejecutarse si no se cumple la condición de entrada.

Generalmente este ciclo es utilizado cuando:

- No se conoce la cantidad de iteraciones.
- Si pueden ejecutarse cero o varias iteraciones.

**Ejemplo:**

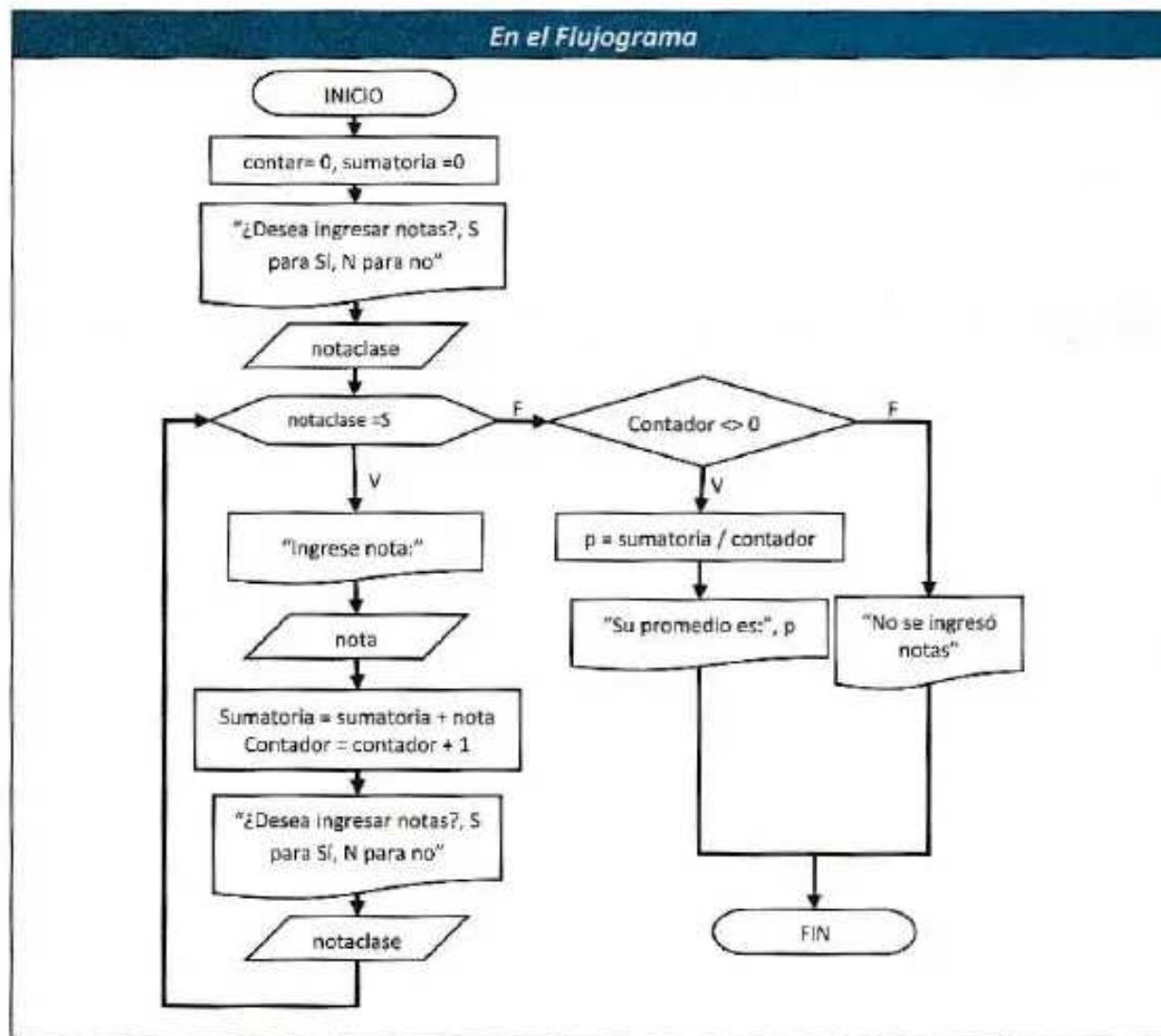
Crear un algoritmo que permita calcular el promedio parcial de las clases de un alumno.

Definición del Problema:

- El problema consiste en que no sabemos cuántas clases lleva el estudiante, pero se quiere calcular su promedio parcial de las clases que introduzca en el programa que estamos realizando, el programa puede realizar la pregunta si dese ingresar una nota, esta respuesta será representada como notaclass, de ese modo si no desea usarlo el programa saldrá y si ingresa que sí, pedirá la nota, acumulará la sumatoria y la dividirá entre el total de notas de las clases ingresadas.

Análisis del Problema:

- Sabemos que se repetirá el ingreso de notas, cada una de las notas se recibirá en la variable nota, el número de notas ingresadas lo almacenaremos en una variable contador que la llamaremos contar; la suma de todas las notas ingresadas la almacenaremos en una variable acumulador que la llamaremos sumatoria. Al dividir sumatoria por contar (sumatoria / contar) obtendremos el promedio que lo almacenaremos en la variable p.

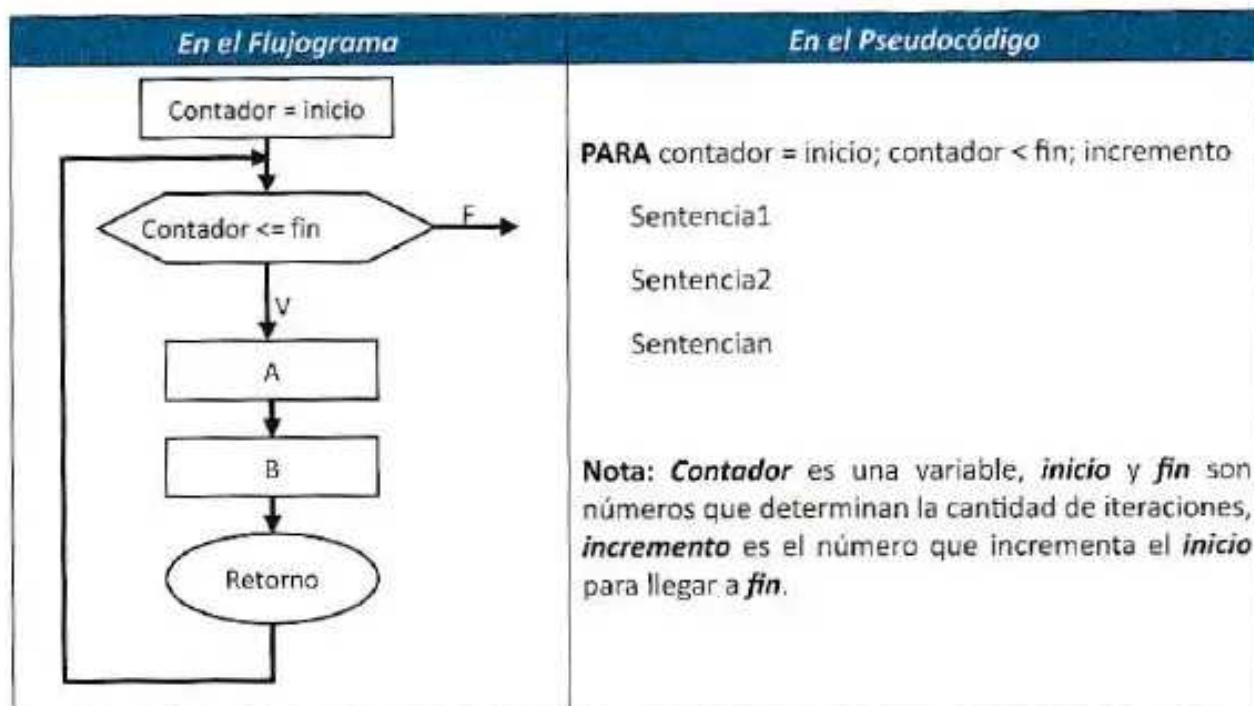
**En el Pseudocódigo**

```

    Inicio
        HAGA contar = 1
        HAGA sumatoria = 0
        ESCRIBIR "¿Desea ingresar notas?, S para Sí, N para no"
        LEER notaclase
        MIENTRAS notaclase = "S" HAGA
            ESCRIBIR "Ingrese Nota"
            LEER nota
            Hacer sumatoria = sumatoria + nota
            Hacer contar = contar + 1
            ESCRIBIR "Desea Ingresar Notas presione S para Sí, N para No"
            Leer notaclase
            Hacer P = sumatoria / contar
            SI P <> 0 ENTONCES
                ESCRIBIR "Su Promedio es:", P
            SINO
                ESCRIBIR "No se ingresó notas"
    Fin

```

Sentencia Para (For): Es una sentencia de control que permite la ejecución de un bloque de instrucciones un determinado número de veces y el número de iteraciones se conoce por anticipado. A diferencia de **While** y **Do While** que pueden ejecutarse indeterminadamente, esta sentencia requiere que conozcamos por anticipado el número de veces que se ejecutan las sentencias del interior del ciclo.



Reglas:

- Las variables de control, valor inicial y valor final deben ser todas del mismo tipo, pero el tipo real no está permitido. Los valores inicial y final pueden ser tanto expresiones.
- Es ilegal modificar el valor de la variable de control, valor inicial y el valor final dentro del bucle.
- Al igual que en las sentencias de selección, las sentencias repetitivas se pueden anidar.

Ejemplo:

Crear un algoritmo que muestre la tabla de multiplicar según el número introducido por el usuario, la tabla se realizará del 1 hasta el 10.

Definición del Problema:

- El problema consiste en mostrar la tabla de multiplicar que el usuario desea.

Análisis del Problema:

- Tenemos como dato de entrada el número introducido por el usuario.
- Como salida tenemos la tabla de multiplicar.

En el Flujograma	En el Pseudocódigo
<pre> graph TD INICIO([INICIO]) --> P1["¿Qué tabla desea visualizar?"] P1 --> T[/T/] T --> C0[Contador = 0] C0 --> D{Contador <= 10} D -- V --> M1[Mult = Contador * T] M1 --> P2["Contador "X", T, " = ", Mult"] P2 --> RET([Retorno]) RET --> FIN([FIN]) D -- F --> FIN </pre>	<p>Inicio</p> <p>ESCRIBIR "Que tabla desea visualizar"</p> <p>LEER n</p> <p>PARA Contador = 0; Contador <= 10; Contador = Contador + 1 HAGA</p> <p>HACER Mult = Contador * T</p> <p>ESCRIBIR Contador "X", T, " = ", Mult</p> <p>Fin</p>

Ejercicio Guiado

Crear un algoritmo que permita leer dos valores distintos y devolver como resultado cuál de ellos es mayor y si son iguales.

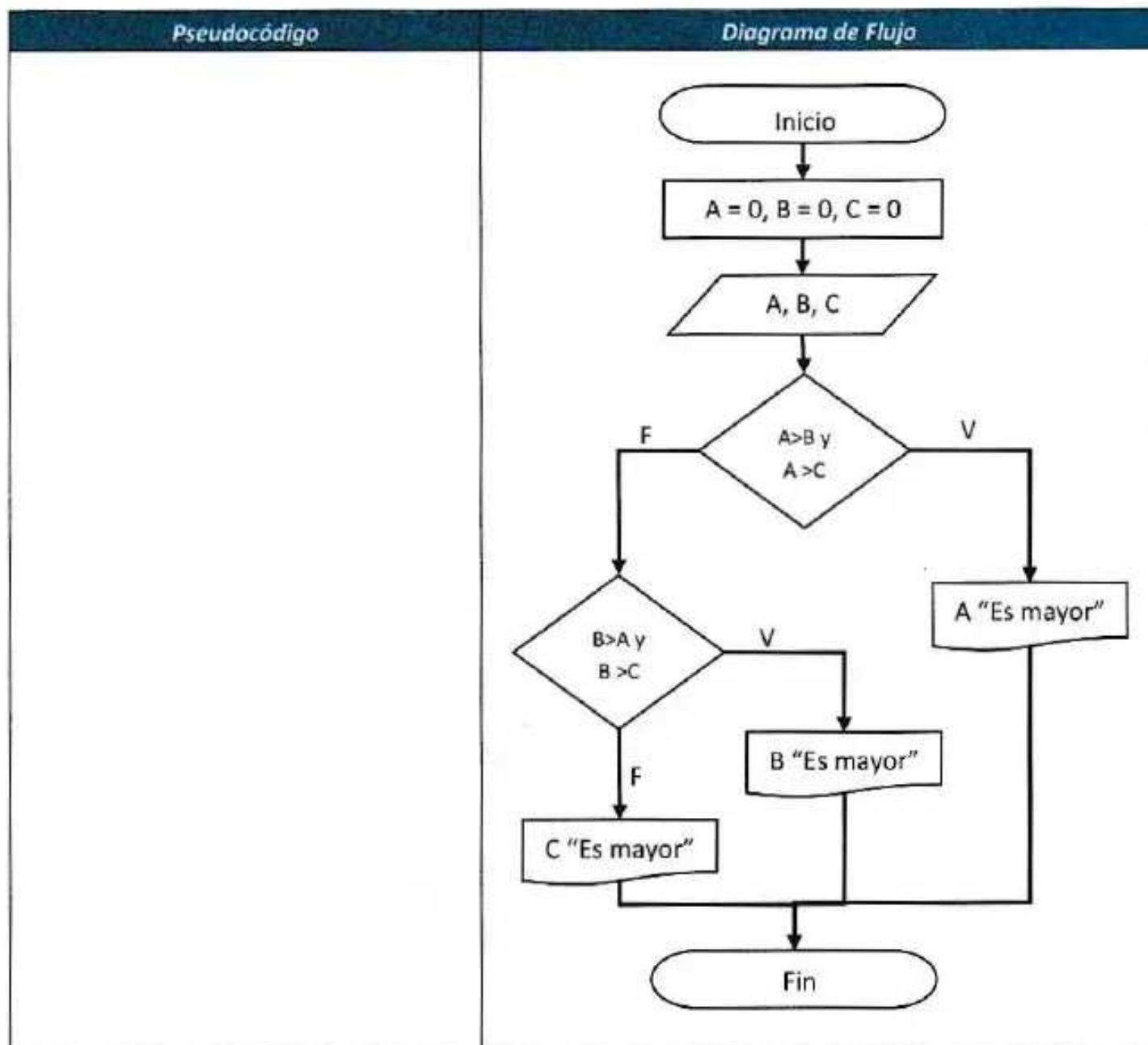
Pseudocódigo	Diagrama de Flujo
<ol style="list-style-type: none"> 1. Inicio 2. Iniciar variables: A = 0, B = 0 3. Solicitar la introducción del primer valor 4. Leer A 5. Solicitar la introducción del segundo valor 6. Leer B 7. Si A = B Entonces Escribir A y B "Son iguales" 8. De lo contrario Evaluar 9. Si A > B Entonces Escribir A, "Es mayor" 10. De lo contrario: Escribir B, "Es mayor" 11. Fin 	<pre> graph TD Inicio([Inicio]) --> A0B0[A = 0, B = 0] A0B0 --> PeticionA["Introduzca el primer valor:"] PeticionA --> AParallelogramo[/A/] AParallelogramo --> PeticionB["Introduzca el segundo valor:"] PeticionB --> BParallelogramo[/B/] BParallelogramo --> CondA_B{A = B} CondA_B -- F --> CondA_B_A_gt_B{A > B} CondA_B -- V --> A_Igual_B["A y B \"Son iguales\""] CondA_B_A_gt_B -- F --> A_Mayor["A \"Es mayor\""] CondA_B_A_gt_B -- V --> B_Mayor["B \"Es mayor\""] A_Mayor --> Fin([Fin]) B_Mayor --> Fin </pre>

Estrategia de Aprendizaje # 7

Instrucciones: Realice cada una de las actividades que se le muestran a continuación:

Actividad 1:

Realice el pseudocódigo basado en el diagrama de flujo que se le muestra:



Actividad 2:

Crear un algoritmo que realice la sumatoria de los números enteros comprendidos entre el 1 y el 10.

Pseudocódigo	Diagrama de Flujo

Programación Modular

Es una estrategia de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible. Un módulo es cada una de las partes de un programa que resuelve uno de los problemas en que se divide el problema complejo original.

La programación modular es una evolución de la programación estructurada que nace como consecuencia de la búsqueda de una alternativa para la solución de problemas demasiados complejos que a través de la programación estructurada llevarían mucho tiempo resolverlos.

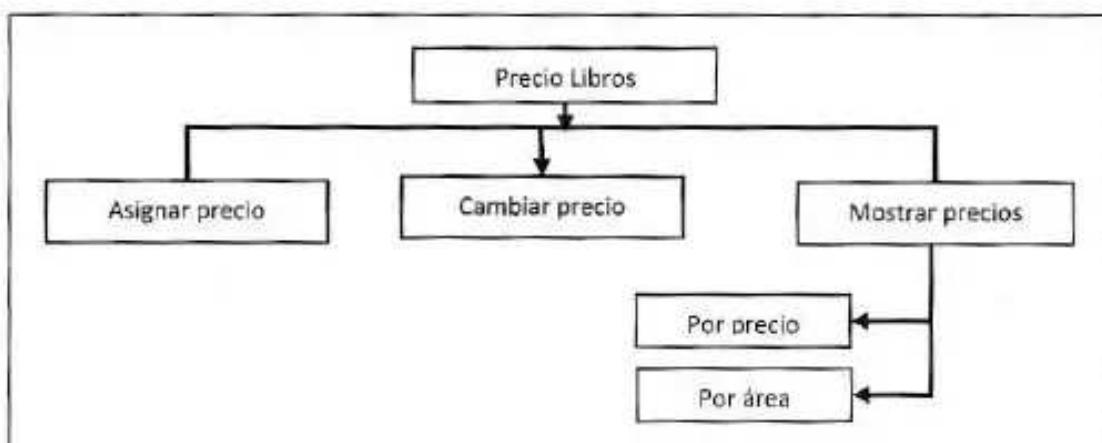
Cada módulo dentro del programa tiene una tarea bien definida, y en algunos casos unos módulos ocupan de otros para poder operar.

Para el programador no es obligatoria la división de un programa en módulos, pero hay que tomar en cuenta los siguientes factores, los módulos:

- Hacen más legible y manejable un programa.
- Simplifican un problema bajo el teorema de divide y vencerás.
- Aumentan su capacidad de ser reutilizado.
- Facilitan el trabajo en equipo.

Ejemplo:

El propietario de Ediciones Fares necesita un programa para gestionar los precios de los libros, el programa debe de ser capaz de asignar precio, cambiar el precio si es necesario, y mostrar el precio en pantalla.



Hemos utilizado la programación modular para degradar el problema principal en sub problemas, por esta acción a este método de programación también se le conoce como **Diseño de Programación Descendente**. La programación descendente comienza en la parte superior con un problema general y se diseñan soluciones específicas (módulos).

Ejemplo:

Crear un programa conversor de medidas de longitud, específicamente de centímetros a pulgadas y viceversa con un menú donde el usuario pueda seleccionar el tipo de conversión que desea.

Definición del Problema:

- Necesitamos un programa capaz de convertir centímetros a pulgadas y pulgadas a centímetros.

Análisis del Problema:

- Como entrada se tendrá:
 - La opción de menú.
 - Los centímetros
 - Las pulgadas

- Como salida tenemos:

• Centímetros (cm) o Pulgadas (pulg)

El análisis de este programa nos refleja que su grado de complejidad es muy alto y debemos de tener cuidado con su algoritmo.

Selección de la Mejor Alternativa:

- Por el grado de complejidad que muestra el problema y el desarrollo del programa utilizaremos programación modular.

Diagrama del Programa Principal:

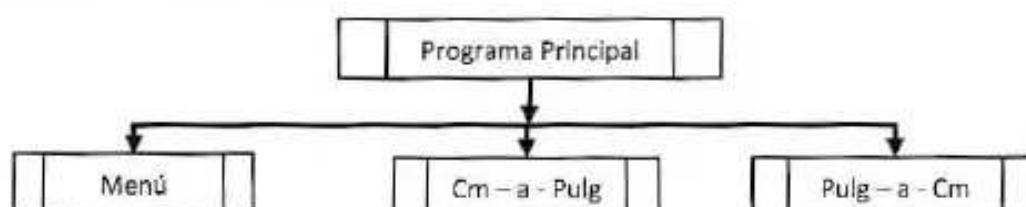
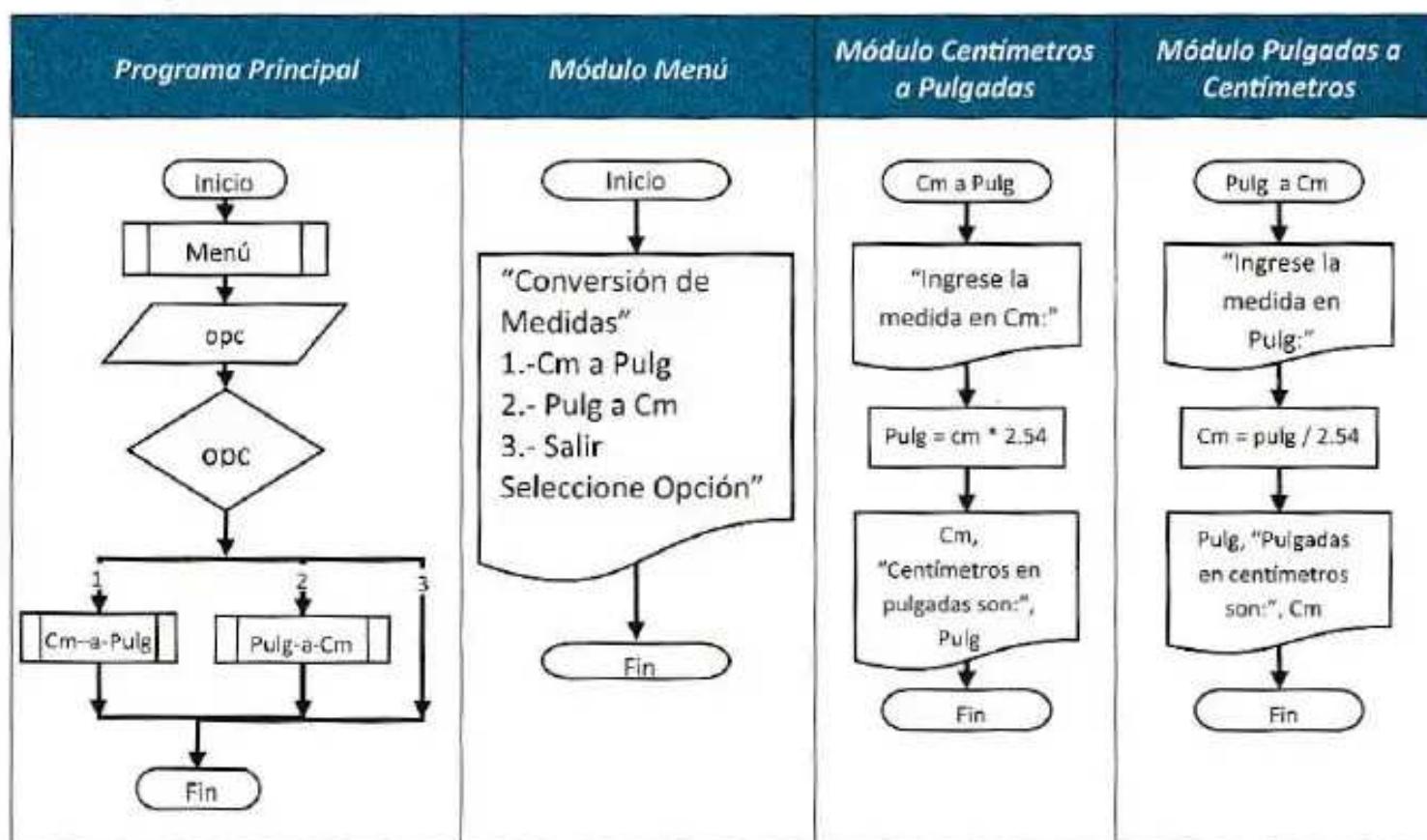


Diagrama Modular:



El proceso de descomposición de un problema en módulos se conoce como **modulación** y a la programación relativa a ellos como **programación modular**.

Los subproblemas o módulos se diseñan con subprogramas, estos a su vez se clasifican en procedimientos y funciones.

Procedimientos y Funciones

Los procedimientos y funciones son la base en la programación modular.

Procedimientos

Un procedimiento es un subprograma o fragmento de código que realiza una tarea específica independientemente del programa en el que se encuentre. Para invocarlo, es decir, para hacer que se ejecute, bastará con escribir su nombre en el cuerpo de otro procedimiento o en el programa principal.

Con los procedimientos se pueden crear algoritmos de ordenación de arrays, de modificación de datos, cálculos paralelos a la aplicación, activación de servicios, etc.

Funciones

Es un fragmento de código o subprograma que realiza una actividad específica devolviendo un valor al proceso que lo llame.

Las funciones y los procedimientos son similares, pero presentan algunas diferencias, entre las cuales tenemos:

- Las funciones sólo retornan un único valor, mientras que los procedimientos pueden devolver cero uno o varios valores. Por lo general, los procedimientos son creados para ejecutar procesos que no devuelvan valores.
- A un nombre de procedimiento no se puede asignar un valor y por consiguiente, ningún tipo está asociado con un nombre de procedimiento.
- Los procedimientos son autónomos, las funciones no.
- Las funciones se refencian utilizando su nombre en una instrucción de asignación o expresión matemática, mientras que un procedimiento se referencia por una llamada.

Transferencia de Información entre Módulos

Parámetro

Son el medio a través del cual podemos expandir el ámbito de variables locales de funciones, hacia otras funciones y además permiten establecer comunicaciones entre funciones. También podemos decir que un parámetro es una variable que puede ser recibida por una rutina o subrutina; esta a su vez, usa los valores asignados a sus argumentos para alterar su comportamiento en tiempo de ejecución. Un argumento representa el valor que se transfiere a un parámetro del procedimiento cuando se llama al **procedimiento**.

Los módulos de acuerdo a la inclusión o no de parámetros se clasifica en:

- **Módulos sin Parámetros:** No existe comunicación entre el programa principal y los módulos o entre dos módulos.
- **Módulos con Parámetros:** Si existe comunicación entre el programa principal y los módulos o entre dos módulos.

La sentencia que contiene el llamado de una función o procedimiento consta de dos partes:

- Identificador o nombre del módulo (nombre del procedimiento o nombre de la función).
- Lista de parámetros.

Ejemplo: NombreSubPrograma(Pa1,Pa2....)

Existen varias formas de pasar un parámetro a una función o a un procedimiento:

- **Por Valor:** Pasar parámetros por valor consiste en copiar el contenido de la variable que queremos pasar en otra dentro del ámbito local de la subrutina, en otras palabras consiste en copiar el contenido de la memoria del argumento que se quiere pasar a otra dirección de memoria correspondiente al argumento dentro del ámbito de la subrutina; por lo que se tendrán dos valores duplicados e independientes, con lo que la modificación de uno no afecta al otro.
- **Por Referencia:** Consiste en proporcionar a la subrutina que se le quiere pasar el argumento la dirección de memoria del dato. En este caso se tiene un único valor referenciado (o apuntado) desde dos puntos diferentes, el programa principal y la subrutina a la que se le pasa el argumento, por lo que cualquier acción sobre el parámetro se realiza sobre el mismo dato en la memoria.
- **Por Valor Resultado:** Se basa en que dentro de la función se trabaja como si los argumentos hubieran sido pasados por valor, pero al acabar la función los valores que tengan los argumentos serán copiados a las variables que pertenezcan.

Variables Locales y Variables Globales

Una variable es un espacio de almacenamiento con un nombre simbólico asociado a un espacio en memoria. Las variables se clasifican según su utilización:

- Locales
- Globales

Variables Locales

Son las variables que están declaradas en el programa principal o módulos y son propias al ámbito de la declaración, en este sentido cada una de estas es distinta a otras variables declaradas en el program

principal o cualquier modulo. Las variables locales solamente van a poder ser manipuladas en la sección donde fueron declaradas.

Variablos Globales

Son aquellas que se declaran fuera del cuerpo de cualquier función, normalmente al principio del programa, después de la definición de los archivos de biblioteca (#include), de la definición de constantes simbólicas y antes de cualquier función. El ámbito de una variable global son todas las funciones que componen el programa, cualquier función puede acceder a las variables para leer y escribir en ellas. Es decir, se puede hacer referencia a su dirección de memoria en cualquier parte del programa.

Ámbito y Visibilidad de un Identificador

La “**visibilidad**” de un identificador determina las partes del programa en las que se puede hacer referencia a él (su “ámbito”). Un identificador está visible sólo en las partes de un programa enmarcado por su “ámbito”, que se puede limitar (en orden de restricción creciente) al archivo, función, bloque o prototipo de función en que aparece. El ámbito de un identificador es la parte del programa en que se puede usar el nombre.

Técnicas para mejorar la Productividad al Programar

1. **Programación Modular:** Es sin duda una de las mejores alternativas al programar y más cuando necesitamos crear un programa con un grado de complejidad muy alto.
2. **Inspecciones Repetitivas:** Es una técnica de programación que permite mejorar la productividad a través de la continua revisión de nuestro programa con esto aseguramos un producto final libre de errores, desde que definimos nuestro problema a solventar hasta que tenemos el ejecutable listo para ser distribuido.
3. **Bibliotecas Virtuales:** Son todas las direcciones web que debemos visitar con frecuencia, que contienen información actualizada de programas, aplicaciones, diseños, diagramas, técnicas, métodos, guías y tutoriales de programación.
4. **Grupos con Programador en Jefe:** Es programar en grupo con un director principal experimentado, también esta técnica mejora la productividad y disminuye errores en el programa final siempre y cuando se dé una interactividad de conocimientos entre los integrantes. El grupo puede incluir otros programadores, analistas y usuarios finales. El programador en jefe es responsable del diseño y codificación de los programas producidos por el grupo.

Anotaciones importantes al final del estudio de la unidad:

Autoevaluación # 1

Tipo Verdadero o Falso

Instrucciones: Escribir en el paréntesis de la derecha una V en caso que considere la proposición como verdadera o una F en caso que la considere falsa, justifique su respuesta en caso de ser falsa.

1. Programa es un conjunto de algoritmos capaces de ser entendidos por un ordenador..... ()

2. Ser eficaz es una características de un algoritmo..... ()

3. Definir el problema es el primer paso para crear un algoritmo..... ()

4. La programación modular es una de las mejores alternativas para programar..... ()

5. La lógica computacional se utiliza en la programación lógica..... ()

6. Las tablas de verdad muestran en forma sistemática los valores de verdad de una proposición..... ()

7. El siguiente ejemplo: *Me siento solo*, es una proposición compuesta..... ()

8. La Negación es una proposición compuesta que resulta de conectar dos proposiciones... ()

9. Las tautologías son vocablos griegos que hacen referencia a la repetición de un mismo pensamiento a través de distintas expresiones..... ()

10. La contradicción es aquella proposición que en todos los casos posibles de su tabla de verdad su valor es siempre de verdad,..... ()

Tipo Enumeración

Instrucciones: Enlistar en forma clara y ordenada lo que continuación se le pide:

1. Cite los tipos de variables utilizadas en un programa con módulos:

a) _____
b) _____

2. Enumere las características de un algoritmo:

a) _____
b) _____
c) _____

3. Enumere los 6 primeros pasos para la solución del problema llamado algoritmo:

a) _____
b) _____
c) _____
d) _____
e) _____
f) _____

4. Enumere 6 ejemplos de tautologías:

a) _____
b) _____
c) _____
d) _____
e) _____
f) _____

Tipo Respuesta Breve

Instrucciones: Responder de forma objetiva las siguientes interrogantes:

1. ¿A qué nos referimos cuando hablamos de ámbito?

2. ¿Cuáles son los 2 tipos de datos comunes más utilizados en la programación?

3. ¿Cuál es la diferencia que existe entre un parámetro o un argumento?

4. ¿Qué es una variable?

5. ¿Qué es una variable local?

6. ¿Cuál es la importancia de las tablas de verdad?

7. ¿A qué llamamos tautologías?

Tipo Práctico

Instrucciones: Resuelva los siguientes ejercicios:

- Crear un algoritmo (diagrama de flujo y pseudocódigo) de un programa que muestre en pantalla los primeros 100 números impares.
- Crear un algoritmo, que presente la conversión de medidas de longitud de metros a kilómetros (m a km).
- Crear un algoritmo para calcular el área de un círculo.
- Crear un algoritmo que calcule el área de trapecios.
- Crear un algoritmo que calcule raíz cuadrada de números enteros.
- Crear un programa que determine el máximo común divisor de dos números utilizando ciclo *For*.
- Crear un programa que muestre en pantalla los números comprendidos entre 31 y 90 utilizando *Hacer Mientras*.
- Crear un programa con menú en dónde el usuario introduzca una medida de longitud especificando si está en centímetros, kilómetros o pulgadas y se realice la conversión a metros utilizando *Case*.
- Crear un algoritmo que realice la sumatoria de los números enteros múltiplos de 5, comprendidos entre el 1 y el 100 utilizando todos los ciclos visto en la unidad.
- Crear un algoritmo que lea los primeros 300 números enteros y determine cuántos de ellos son impares.
- Determinar si las siguientes proposiciones son Negación, Conjunción, Disyunción, Bicondicional:

 - Juan no conversa _____
 - Yo trabajo si y sólo si me paga la empresa _____
 - Vino Rosa de España y me trajo regalos _____
 - Ricardo juega baloncesto o Rosa juega fútbol _____

- Desarrollar un algoritmo que permita leer un valor entero positivo N y muestre un mensaje diciendo si es primo o no.
- Elaborar la tabla de verdad de: $p \Rightarrow (q \wedge r) \vee (r \wedge \neg q) \vee r$

6 Un estudiante realiza cuatro exámenes durante el semestre. Realice el pseudocódigo y el diagrama de flujo que representen el algoritmo correspondiente para obtener el promedio de las calificaciones obtenidas.

6 Determinar si las siguientes proposiciones son simples o compuestas:

- () El perro durmió _____
- () Fui al banco, pero el banco estaba cerrado _____
- () Los lectores de este libro son jóvenes o adultos _____
- () Si sacas excelentes calificaciones entonces te regalo un auto _____
- () El sol está brillando _____
- () María juega fútbol _____
- () El gato estaba dormido pero el hambre lo despertó _____



UNIDAD



PROGRAMACIÓN VISUAL C++

Estructurada y Orientada a Objetos

Competencias de Unidad:

- ◀ Definir el término programación.
- ◀ Establecer las generalidades de una programación estructurada y orientada a objetos.
- ◀ Depurar procesos resultantes.

Expectativas de Logro:

- ◀ Identifican los tipos de programación para definir su aplicación en la búsqueda de alternativas para desarrollar un programa.
- ◀ Analizan, diseñan y resuelven problemas a través de la programación.

Elementos de Competencia:

El estudiante es competente cuando:

- ◀ Comprende el lenguaje de programación estructurada y orientada a objetos.
- ◀ Crea programas utilizando Visual C++.

Contenidos:

- ◀ Codificación
- ◀ Generalidades de C++
- ◀ Visual C++
- ◀ Sentencias o Estructuras de Control en C++

Saberes Previos:

- En el espacio de aprendizaje los estudiantes se reúnen en equipos para realizar un **Diagnóstico Inicial**, respondiendo las siguientes interrogantes:
 - ¿Cuál es la importancia de programar?
 - ¿Qué es un lenguaje de programación?
 - ¿Para qué nos sirve C++?
 - ¿Qué entiende por codificar?



Construcción de Saberes:

- En el Laboratorio de Cómputo el docente explica de forma **magistral** el contenido de la unidad, realizando paso a paso los ejemplos sugeridos en la misma. A su vez, aplica la **Técnica: Dibujando Nuestros Conocimientos**.

Codificación

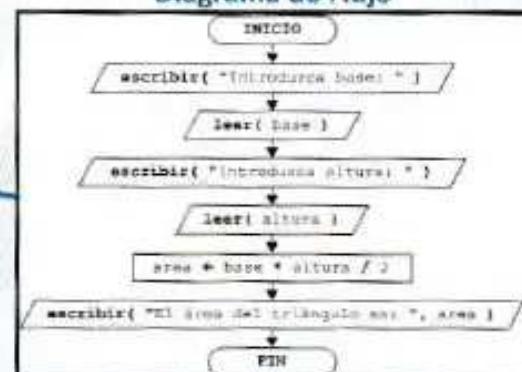
```
Procedimiento Ordenar (L)
//Comentario: L = (L1, L2, ..., Ln) es una lista con n elementos
k ← 1;
Repetir
    intercambio ← Falso
    k ← k + 1;
    Para i ← 1 Hasta n - k Con Paso 1 Hacer
        Si Li > Li+1 Entonces
            intercambiar (Li, Li+1)
            intercambio ← Verdadero
        FinSi
    FinPara
    Hasta Que intercambio = Falso;
FinProcedimiento
```



Pseudocódigo

```
Proceso Condicionable
    Definir Número como real;
    Leer Número;
    Si Número > 6 entonces
        Escribir "Aprobó la materia";
    Sino
        Escribir "No aprobó la materia";
    FinSi;
    Si (Número >= 5) entonces
        Si (Número <= 3) Entonces
            Escribir "Puede ganar una beca";
        FinSi;
    FinSi;
FinProceso
```

Diagrama de Flujo



Explicación de la Técnica:

Es una técnica de recreación que durante la explicación magistral del docente, el estudiante va construyendo sus conocimientos. De esta manera los discentes hacen una presentación creativa empleando dibujos y textos cortos. El docente al explicar cada tópico de la unidad va supervisando que el estudiante a su vez vaya construyendo aprendizajes significativos.

Materiales:

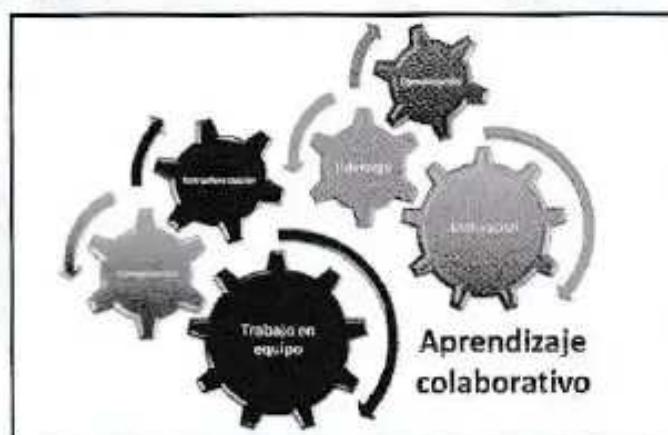
- ④ Hojas bond
- ④ Lápices
- ④ Colores

El procedimiento para el desarrollo esta técnica es el siguiente:

- ④ Prestar atención de cada tópico explicado y contemplado en la unidad.
- ④ Identificar las ideas de cada tema.
- ④ Dibujar las ideas, agregándoles un texto escrito de acuerdo a convicción personal del estudiante.
- ④ Una vez teniendo todo el esquema por escrito, realizarlo en la computadora.
- ④ Reunirse con sus demás compañeros para socializar el trabajo.
- ④ Elaborar conclusiones en grupo de todos los trabajos elaborados.

Consolidación de Saberes:

- El estudiante, desarrolla las **Estrategias de Aprendizaje** y **Autoevaluación** incluidas en el interior y final de la unidad, respectivamente.



Valoremos lo Aprendido:

- Elabora un programa a través de C++.

Discusión Dirigida

Instrucciones: El docente y estudiante discuten la unidad en conjunto, donde el discente hace las anotaciones resultante de la misma y a su vez escribe los aportes del profesor en el siguiente cuadro:

APORTE DEL DOCENTE

CONCLUSIONES DEL ESTUDIANTE



Codificación

Es la conversión de un algoritmo en programa. Es la etapa de escritura de la solución de un problema (diagrama de flujo y pseudocódigo) en un código reconocible para la computadora a través de un *lenguaje de programación*.

Antes, la codificación era la etapa más difícil para un programador, pero con la invención de los lenguajes de programación de alto nivel su dificultad fue reducida. En la actualidad, contamos con poderosos lenguajes de programación que nos facilitan mucho la tarea de programar, nos enfocaremos en el *Lenguaje C++*.

Lenguaje C++

Es un lenguaje que permite programar desde sistemas operativos, compiladores, aplicaciones de bases de datos, procesadores de texto hasta juegos, etc.

Reseña Histórica C++

C++ nació como una extensión del lenguaje C, para conseguir aunar la eficiencia del lenguaje C con las ventajas del modelo orientado a objetos.

El primer paso hacia C++ se produjo en 1980, cuando se presentó un primer lenguaje llamado **C with classes**, descrito por Bjarne Stroustrup. En 1983/84 se rediseñó este lenguaje, pasando a llamarse **C++** (el nombre proviene del operador incremento de C, `++`, para indicar que es una evolución). Tras pequeños refinamientos, en 1985 se puso a disposición de todo el mundo y se documentó en el libro de Bjarne Stroustrup, "*The C++ Programming Language*".

Además, C++ es compatible con C, es decir, todo lo que puede hacerse en C se puede hacer también en C++, por lo tanto, cualquier código C puede tratarse con un compilador de C++. Por esta razón, un gran porcentaje de usuarios de C++ lo emplean simplemente como un C más potente.

Estructura Básica de un Programa en C++

Todo programa creado en C++ debe contener las siguientes partes:

1. Un contenido descriptivo del programa.
2. Escribir cada una de las librerías que se utilizarán (encabezado).
3. Escribir la función principal.
4. Declaración de constantes, variables, tipos de datos y sus clases.
5. Cuerpo del programa.

Comentario Descriptivo del Programa

Son anotaciones legibles hechas en el código fuente de un programa significativas para el programador porque registran aspectos importantes del programa principal y de sus módulos. Un comentario puede hacer que el usuario y programador del software entienda con mayor claridad el código del mismo. Los comentarios son añadidos para que el código fuente de un programa sea más entendible a la hora de crear el programa, ofrecer mantenimiento y reutilizarlo. Es importante tener claro, que los comentarios también necesitan mantenimiento igual que el código, un comentario preciso y conciso es más fácil de mantener que un comentario largo, redundante, repetitivo y complejo. Los comentarios no son tomados en cuenta por el compilador.

Para colocar un comentario en C++ primeramente debemos conocer cuantas líneas tendrá.

- Si el comentario solamente tiene una linea, este debe iniciar con dos plecas contiguas “//”.
Ejemplo: //Comentario de una línea
- Si el comentario tiene más de una linea entonces este debe empezar con pleca asterisco “/*”, seguido del comentario y finalizar con asterisco pleca “*/”.
Ejemplo: /* En este comentario puede escribir lo que
desea para aclarar el código escrito */

Librerías que se utilizarán (Encabezado)

Las librerías son paquetes o colecciones de clases y funciones ya definidas en el lenguaje que permiten dar solución a problemas comunes y que generalmente requieren de acciones genéricas.

Todos los compiladores C y C++ disponen de ciertas librerías de funciones estándar que facilitan el acceso a la pantalla, al teclado, a los discos, la manipulación de cadenas, y muchas otras cosas, de uso corriente. Para utilizar una librería se debe utilizar la **directiva #include**.

Directiva #include

Son una serie de instrucciones en un programa que comienzan con el símbolo numeral “#” utilizados para crear programas fuentes que puedan ser modificados y compilados fácilmente en diferentes entornos de ejecución.

La directiva **#include** es utilizada para incluir la declaración de otro fichero en la compilación, cuando el procesador encuentra una línea **#include** reemplaza esta línea por el fichero incluido.

Para tener acceso a una función predefinida en una librería, se deberá escribir en las primeras líneas de un programa la directiva **#include** seguida del **nombre del fichero** que contiene la función que se utilizará.

Sintaxis

```
#include<Nombre de la Librería>
```

C++ cuenta con las siguientes librerías estándar:

- ***iostream.h***: Utilizada para entrada y salida de datos.
- ***ctype.h***: Es una librería usada para operaciones básicas con caracteres.
- ***string.h***: Utilizada por sus plantillas estándares para trabajar con cadena de caracteres.
- ***math.h***: Utilizada para la función de operaciones matemáticas.
- ***fstream***: Utilizada para la entrada y salida de datos.
- ***stdlib.h***: Utilizada para controles de procesos y sobre todo, para almacenamientos dinámicos.
- ***assert.h***: Usada para comprobar condiciones en el programa.
- ***setjmp.h***: Utilizada para indicar el salto de función en el flujo.
- ***memory***: Utilizada para las gestiones de almacenamiento de memoria.
- ***signal.h***: Utilizada para reportar un evento dentro del programa.
- ***time.h***: Utilizada en la gestión de los formatos fecha y hora.
- ***list***: Utilizada para la creación de listas enlazadas.
- ***numeric***: Es la librería numérica, utilizada para operaciones numéricas.
- ***forward_list***: Utilizada para enlazar listas simples.

Estas librerías estándar son las más comunes en C++, ya que en el proceso de programar nos encontraremos con diversidad de exigencias por parte del usuario y nosotros debemos de tener la preparación competitiva de saber desde un principio como faremos para crear el programa que el cliente desea y sobretodo; saber que herramientas nos proporciona nuestro lenguaje de programación y cuales se adaptan a nuestra necesidad.

Función principal

En C++ todo programa debe tener una función que encierre todas las sentencias del programa, para programas sencillos se utilizará la ***función main ()***. La función principal indica dónde comienza y finaliza el programa.

Sintaxis:

```
Int main ()
{
    Bloque de instrucciones
}
```

Declaraciones

Algunos elementos programables tienen que declararse antes de usarse en el programa para reservar el espacio de memoria o la categoría a las que pertenecen.

Las declaraciones son definiciones de variables o constantes que serán utilizadas por cualquiera de las funciones definidas en el programa.

Sintaxis para la declaración de una variable:

Tipo var1, var2,....varn;

También se puede declarar y asignar un valor:

Tipo var1 = valor

Cuerpo del programa

Son las líneas de programación con las cuales se hace cumplir el objetivo del programa.

Generalidades de C++**Identificadores**

En la mayoría de los programas de computadora, es necesario manejar datos de entrada o de salida, los cuales necesitan almacenarse en la memoria principal de la computadora en tiempo de ejecución. Para poder manipular los datos, se necesita acceso a las localidades de memoria donde se encuentran almacenados; esto se logra por medio de los nombres de los datos o *identificadores*.

Los identificadores también se utilizan para nombres de los programas, procedimientos, de las funciones, etiquetas, constantes y variables.

Un identificador es un conjunto de caracteres alfanuméricos de cualquier longitud que sirve para identificar las entidades (programas, procedimientos, funciones, etiquetas, constantes y variables) del programa.

Reglas para formar los identificadores en C++:

1. Un identificador puede estar formado por caracteres alfabéticos, numéricos y el carácter de subrayado (_).
2. Deben comenzar con un carácter alfabético o el carácter de subrayado.
3. Puede ser de cualquier longitud (sólo los 31 primeros caracteres son significativos).
4. Se hace distinción entre mayúsculas y minúsculas (dependiendo el compilador).
5. No se permite el uso de los **Identificadores Reservados** en los nombres de variables, constantes, programas o sub-programas.

6. Los identificadores son únicos, no se pueden usar para dos cosas diferentes dentro del mismo ámbito.
7. Las letras mayúsculas y minúsculas son diferentes para C++.

Ejemplos de identificadores válidos:

- Nombre
- Edad
- Dirección
- Sum2
- _arroba
- Dias
- Resta
- Nom_trab
- Fares25

Ejemplos de identificadores no válidos:

Identificadores no válidos	Justificación
Num&Dias	Carácter & no válido
Edic fares	Contiene un blanco
nazareth@123	Carácter @ no válido
14abc	No inicia con una letra
Num.1	El punto no está permitido
Cin	Palabra reservada
return	Palabra reservada

A continuación, se menciona algunas palabras reservadas de C++, las cuales no pueden utilizarse como identificadores; no están todas las palabras reservadas sólo las más comunes. Si se intenta utilizar una palabra como identificador a una entidad propia y se compila, se mostraría un mensaje de error.

adjustfield	Extern	ofstream	srand	do
asm	False	operator	static	double
auto	Fill	ostream	static_cast	dynamic_cast
basefield	Fixed	precision	stream	else
bool	Flags	private	streambuf	endl
break	Float	protected	struct	enum
case	floatfield	public	switch	explicit
catch	Flush	rand	template	istream

cerr	For	RAND_MAX	this	Left
char	Friend	register	throw	Long
cin	fstream	register	time	Mutable
class	Goto	reinterpret_cast	true	Namespace
clock	Hex	return	try	New
const	If	rigth	typedef	Oct
const_cast	ifstream	scientific	typeid	short
continue	Inline	setf	typename	showbase
cout	Int	setfill	union	showpoint
dec	internal	setiosflag	unitbuf	showpos
default	los	setprecision	unsetf	signed
delete	ostream	setw	unsigned	sizeof

Tipos de Datos y Variables

Una variable es un espacio de memoria reservado en el ordenador para contener valores que pueden cambiar durante la ejecución de un programa. Cada variable tiene un tipo de datos asignado correspondiente al tipo de dato que almacenará, podemos decir, que los tipos de datos determinan cómo se manipulará la información contenida en esas variables.

Tipos de Datos

Los tipos de datos se clasifican en primitivos y derivados.

Los tipos de datos primitivos en C++ son:

- Numéricos enteros
- Numéricos reales
- Tipo lógico
- Tipo carácter

Tipos de Datos Numéricos Enteros

El tipo de dato numérico es un subconjunto finito de los números enteros del mundo real. Pueden ser positivos o negativos.

Tipo de Dato	Descripción	Número de bytes típico	Rango
short	Entero corto	2	-32768 a 32767
int	Entero	4	-2147483648 a +2147483647
long	Entero largo	4	-2147483648 a +2147483647
char	Carácter	1	-128 a 127

Con los tipos enteros pueden utilizarse los calificadores ***signed*** y ***unsigned***. Estos calificadores indican si el número tiene signo o no. Si se usan solos sin indicar el tipo de dato, se asume que son de tipo ***int***.

Por ejemplo, las siguientes declaraciones son equivalentes:

`unsigned int p;` equivale a: `unsigned x;`
`signed int a;` es equivalente a escribir: `int a;`

Tipos de Datos Numéricos Reales

El tipo de dato numérico real es un subconjunto finito de los números reales. Pueden ser positivos o negativos.

Tipo de Dato	Descripción	Número de bytes típico	Rango
float	Real (Número en coma flotante)	4	Positivos: 3.4E-38 a 3.4E38 Negativos: -3.4E-38 a -3.4E38
double	Real Doble (Número en coma flotante de doble precisión)	8	Positivos: 1.7E-308 a 1.7E308 Negativos: -1.7E-308 a -1.7E308
long double	Real Doble Largo	10	Positivos: 3.4E-4932 a 1.1E4932 Negativos: -3.4E-4932 a -1.1E4932

Tipo Lógico

Los datos de este tipo sólo pueden contener dos valores: ***true*** o ***false*** (verdadero o falso).

Si se muestran como enteros, el valor ***true*** toma el valor 1 y ***false*** el valor 0.

Tipo de Dato	Descripción	Número de bytes típico	Rango
bool	Dato de tipo lógico	1	0, 1

Tipo Carácter Extendido

Este tipo se utiliza para representar caracteres **UNICODE**. Utiliza 2 bytes a diferencia del tipo char que sólo utiliza 1.

Tipo de Dato	Descripción	Número de bytes típico	Rango
wchar_t	Carácter Unicode	2	0 a 65535

Operadores en C++

Un operador es un elemento de programa que se aplica a uno o varios operandos en una expresión o instrucción. Los operadores que requieren un operando, como el operador de incremento se conocen como **operadores unarios**. Los operadores que requieren dos operandos, como los operadores aritméticos (+, -, *, /) se conocen como **operadores binarios**.

Existen 6 tipos de operadores según su función y son: aritméticos, relacionales, de asignación, lógicos, dirección y de manejo de bits.

Operadores Aritméticos

Son los operadores que se utilizan para realizar cálculos y operaciones con números reales y punteros, los operadores más comunes son:

Operador	Acción	Ejemplo	Resultado
-	Resta	X = 10 - 4	X vale 6
+	Suma	X = 10 + 4	X vale 14
*	Multiplicación	X = 10 * 4	X vale 40
/	División	X = 10 / 4	X vale 2.5
%	Módulo	X = 10 % 4	X vale 2
--	Decremento	X = 1; X--	X vale 0
++	Incremento	X = 1; X++	X vale 2

La operación módulo se refiere a obtener el residuo de la división, de modo que al dividir 30 entre 4 tendremos como resultado 7 y como residuo 2, por tanto 30 % 4 corresponde al 2 que es el valor que sobra de la división exacta.

Es de suma importancia tomar en cuenta, que los operadores de incremento y decremento realizan su operación de acuerdo a la ubicación con respecto a la variable. Si el operador precede a la variable, se conoce como **pre-incremento o pre-decremento** y se dice que el operador está en su **forma prefija**. Por el contrario, si el operador es posterior a la variable se encuentra en la forma **posfija** y se le llama **post-incremento o pos-decremento** según el caso.

Cuando un operador de incremento o decremento precede a su variable, se llevará a cabo la operación de incremento o de decremento antes de utilizar el valor del operando, *por ejemplo*:

```
int x,y;
x = 2015;
y = ++x;
/* x e y valen 2016 */
```

En el caso de los post-incrementos y post-decrementos pasa lo contrario; se utilizará el valor actual del operando y luego se efectuará la operación de incremento o decremento, *por ejemplo*:

```
int x,y
x = 2004;
y = x++;
/* y vale 2004 y x vale 2005 */
```

Operadores Relacionales (Binarios Lógicos)

Se utilizan para comprobar la veracidad o falsedad de determinadas propuestas de relación (respuestas a preguntas). Las expresiones que los contienen se denominan **expresiones relacionales**. Aceptan diversos tipos de argumentos y el resultado (respuesta a la pregunta), es siempre del tipo verdadero/falso; es decir, producen un **resultado booleano**.

Si la propuesta es cierta, el resultado es **true** (un valor distinto de cero), si es falsa será **false** (cero).

Operador	Acción	Ejemplo (x=10, y= 5)	Resultado
<	Menor que	X < Y	Falso
>	Mayor que	X > Y	Verdadero
<=	Menor o igual que	X <= Y	Falso
>=	Mayor o igual que	X >= Y	Verdadero
==	Igual que	X == Y	Falso
!=	Diferente	X != Y	Verdadero

Cualquiera que sea el tipo de los operandos, por definición, un operador relacional, produce un bool (true o false) como resultado, aunque en determinadas circunstancias puede producirse una conversión automática de tipo a valores int (1 si la expresión es cierta y 0 si es falsa).

Operadores Lógicos

Los operadores lógicos producen un resultado booleano y se utilizan para cambiar el valor de verdad de una o más expresiones. Los operadores lógicos son tres, dos de ellos son binarios y el último (negación) es unario.

Operador	Acción	Ejemplo	Resultado
&&	And Lógico	P && Q	Sí P y Q son verdaderos se obtiene verdadero
	Or Lógico	P B	Devuelve verdadero si uno o ambos son verdaderos
!	Negación Lógica	!A	Negación de A

Operadores de Asignación

A continuación, se presenta una tabla con operadores de asignación y su respectivo ejemplo asumiendo que $a=2$, $b=4$, $m=6$, $n=3$:

Símbolo	Uso	Descripción	Resultado
=	$a=b$	Asigna el valor de b en a	$a=3$
=	$a=b$	Multiplica a*b y asigna el resultado en a	$a=6$
/=	$a/=b$	Divide a/b y asigna el resultado en a	$a=2$
+=	$a+=b$	Suma a y b y asigna el resultado en a	$a=6$
-=	$a-=b$	Resta a y b y asigna el resultado en a	$a=1-2$
%=	$a\%b$	Divide a/b y asigna el residuo en a	$a=0$

Forma abreviada y no abreviada de los operadores de asignación

Símbolo	Abreviado	No abreviado	Resultado
=	$m=n$	$m=m*n$	$m=18$
/=	$m/=n$	$m=m/n$	$m=2$
+=	$m+=n$	$m=m+n$	$m=9$
-=	$m-=n$	$m=m-n$	$m=3$
%=	$m\%n$	$M=m \% n$	$m=0$

Constantes

Son los datos almacenados que no cambiarán su valor en ningún punto del proceso del programa. Para declarar una constante, utilizamos la palabra reservada **const**, que nos indica que el valor de la variable no se puede modificar: **const int x = 10;**

Instrucciones

Una instrucción es una línea de código perteneciente al programa, en C++ cada instrucción debe terminar con un punto y coma ";". A una instrucción también se le llama **enunciado, sentencia o estatuto**.

Bloque de Instrucciones

Representa un conjunto de instrucciones simples, en C++ para indicar dónde comienzan y finalizan las instrucciones, estas deben escribirse entre corchetes.

Ejemplo:

```
{
    Sentencia 1;
    Sentencia 2;
    Sentencia n;
}
```

Secuencias de Escape

Una secuencia de escape es un conjunto de constantes establecidas en el lenguaje, que son interpretadas con algún fin. Las secuencias de escape por lo general, se utilizan para especificar acciones como retornos de carro y movimientos de tabulación en terminales e impresoras. También se emplean para proporcionar representaciones literales de caracteres no imprimibles y de caracteres que normalmente tienen significados especiales, como las comillas dobles (""). En la siguiente tabla se muestra una lista de las secuencias de escape más utilizadas den C++:

Secuencia de Escape	Representa	Secuencia de Escape	Representa
\a	Alarma (alerta)	\	Barra diagonal inversa
\b	Retroceso de espacio	\?	Signo de interrogación literal
\f	Avance de página	\"	Doble comillas
\n	Retorno de cursos y avance de línea	\000	Caracter ASCII en notación octal
\r	Retorno de carro	\xhh	Caracter ASCII en notación hexadecimal
\t	Tabulación horizontal	\0	Cero, nulo
\v	Tabulación vertical		

Entrada y Salida de Datos en C++

Son el conjunto de instrucciones que permiten leer datos de entrada de la computadora por lo general del teclado, generan una orden y luego despliegan en pantalla los resultados que el usuario desea. Desde

el punto de vista físico, las instrucciones de entrada y salida le ayudan al programa a comunicarse con un periférico de la computadora (impresora, disco duro etc.), este aspecto es sumamente importante para el usuario ya que le permite interactuar con el programa y satisfacer sus necesidades.

A continuación, estudiaremos las librerías más comunes encargadas de entrada y salida de datos:

Para Salida

1. **Librería iostream:** Es una librería utilizada para operaciones de entrada y salida de datos.

La librería iostream cuenta con los siguientes objetos:

- **Cin:** Flujo de entrada
- **Cout:** Flujo de salida

Todos los objetos derivados de iostream hacen parte del espacio de nombres std.

Para que en pantalla se escriba una cadena de caracteres es necesaria la palabra reservada **Cout**, luego el operador <<, comillas y a continuación la salida que se espera en pantalla con la secuencia de escape.

Std::cout<< "cadena_salida – secuencia de escape";

Ejemplos:

- Crear un programa que imprima en pantalla la frase: Hola Mundo:

• Std::cout<< "Hola Mundo \n";

Imprime Hola Mundo y \n deja el cursor al principio de una línea nueva.

• Std::cout<< "Hola Mundo \t";

Imprime Hola Mundo y \t deja el cursor en la siguiente tabulación.

• Std::cout<< "Hola Mundo \r";

Imprime solamente Hola Mundo, \r retorna el cursor al inicio de la línea actual.

• Std::cout<< "Hola Mundo \\";

Imprime Hola Mundo \\.

• Std::cout<< "Hola Mundo \\";

Imprime solamente Hola Mundo.

2. **Librería stdio.h:** Es otra librería también utilizada para operaciones de entrada y salida.

Nos centraremos en la función de salida **printf ()**, que envía a la pantalla una cadena de texto igual que cout de iostream, y su sintaxis es:

Printf ("cadena","reemplazo1, reemplazo2")

Donde:

cadena: Es la línea de texto a mostrar en pantalla, cadena puede llevar secuencias de escape o comodines de formato para números, los cuales serán reemplazados por reemplazo1, reemplazo2 etc.

Ejemplos:

- Crear un programa que imprima en pantalla la linea de texto: **Hola Mundo**, utilizando printf de la librería stdio.h.

La sentencia quedaría así:

Printf ("Hola Mundo");

- Según lo antes visto mostrar en pantalla el valor de una variable entera en donde: **x = 50**.

Entonces: int x; x = 50;

Printf ("%d", x);

En pantalla muestra el número 50 que es el valor de la variable en este caso x.

Comodines de Formato para Números

Son las especificaciones de conversión utilizados para enviar a la salida o pantalla los valores de las variables.

Los comodines de formato para números más usados son:

- **%i:** Para variables tipo int.
- **%d:** Para variables tipo int.
- **%c:** Para variables tipo char.
- **%f:** Para variables tipo float.
- **%lf:** Para variables tipo double.
- **%s:** Para variables tipo string.
- **%E:** Para notación científica.

Para Entrada

Para que un programa en C++ tenga la capacidad de capturar datos desde un teclado es necesario la palabra reservada **cin**, de la librería **iostream** que pertenece al espacio de nombre **std**, luego del operador **>>** y a continuación la variable donde se almacenará el mismo valor.

Std:::cin>> identificador;

El identificador debe de ser una variable.

Ejemplo:

La siguiente sentencia captura el valor ingresado desde el teclado en una variable en el cual el identificador es "tecla":

```
Std:::cin >> tecla
```

También podemos usar la librería stdio.h, en donde la sentencia a utilizar es **scanf()**, y su sintaxis es:

```
Scanf("comodin", identificador)
```

Tomando la variable tecla del ejemplo anterior y asumiendo que se ingresa un carácter quedaría entre comillas el comodín de formato separado por una coma la variable:

```
Scanf("%c", tecla)
```

Visual C++

Es un compilador de C y C++ que permite la creación de programas estructurando gran variedad de ficheros esenciales para el mismo. Básicamente, este compilador engloba el desarrollo de aplicaciones en C, C++ y C++ CLI en el entorno Windows.

Visual C++ también proporciona bibliotecas de Windows (WinApi), las bibliotecas Microsoft Foundation Classes que son un conjunto de clases interconectadas por múltiples relaciones de herencia, y contiene el entorno de desarrollo para punto NET (.NET Framework). Visual C++ incluye las bibliotecas propias de cada versión del sistema operativo junto con sus sockets. Al igual que otros compiladores, se le pueden añadir nuevas bibliotecas.

En sus versiones más recientes, viene integrado en el entorno Visual Studio también desarrollada por Microsoft que incluye un paquete de herramientas utilizadas para el desarrollo de programas, sitios y aplicaciones web. Actualmente, su versión más actualizada viene en el entorno **Visual Studio Professional 2015**.

Cuenta con una versión gratuita que se puede descargar desde el sitio de Microsoft llamada **Microsoft Visual Studio Community Edition 2015**. En la versión 2015 se han logrado cambios y mejoras importantes:

- ➊ Plantillas de proyectos para el desarrollo de aplicaciones en Android.
- ➋ Plantillas de proyectos para el desarrollo de algunas aplicaciones en iOS.
- ➌ Mejoras a la hora de diagnosticar resultados.
- ➍ Mejoras en la productividad.
- ➎ Mejoras considerables en el tiempo de compilación.

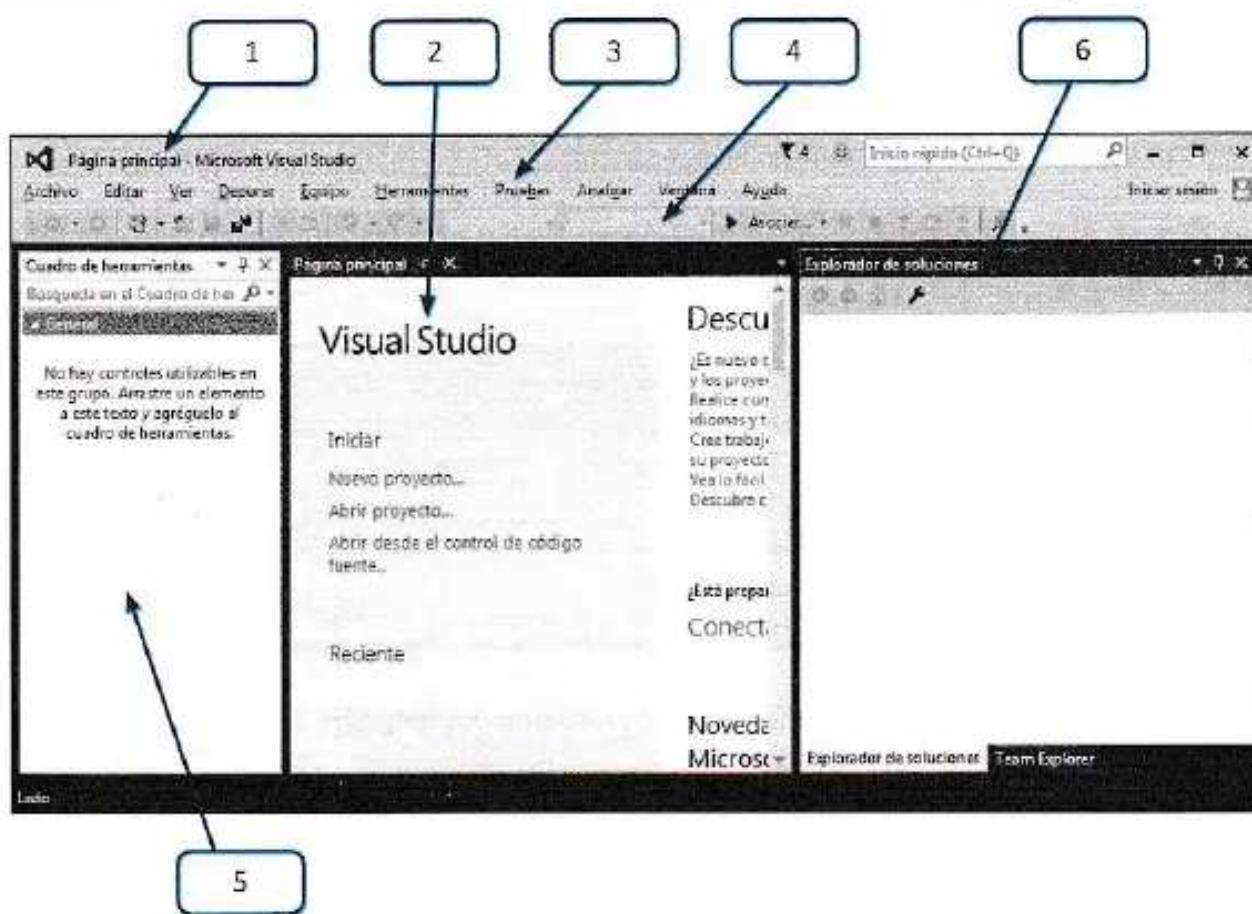
Entorno de Desarrollo Integrado

Visual C++, Visual C#, Visual F# y Visual Basic comparten un mismo entorno de desarrollo (IDE), que permite el uso compartido de herramientas facilitando la creación de aplicaciones en diferentes lenguajes.

Al ingresar a Visual Studio, inicialmente se mostrará la pantalla de Bienvenida de la siguiente manera:



Seguidamente, se mostrará el entorno o pantalla principal de Visual Studio así:



- Barra de Título:** Muestra el nombre del proyecto y el nombre del programa que se está utilizando.
- Página Principal:** Esta página permite crear o tener acceso a proyectos recientes, tener acceso a las próximas publicaciones de productos, así como leer artículos recientes sobre programación.
- Barra de Menú:** Contiene los accesos a cada una de las opciones que Visual Studio. Para acceder cada una de las opciones basta con llevar el mouse y presionar encima del nombre de cualquiera de las opciones o presionar simultáneamente la tecla **Alt** junto con la letra que la opción tenga

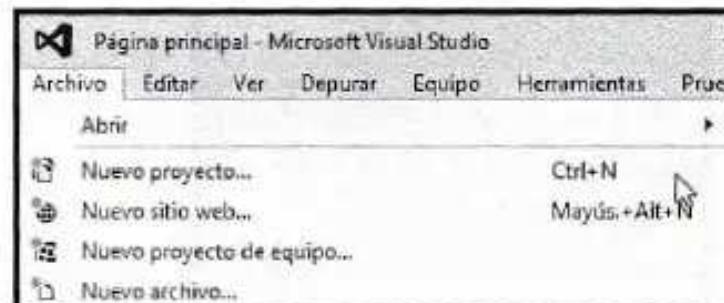
subrayada. Si se quiere acceder al menú de herramientas basta con presionar **Alt + H** y se desplegarán todas las opciones de este menú en la pantalla.

4. **Barra de Herramientas:** Está compuesta por botones que proporcionan atajos a las principales opciones que presentan los menús. Los botones son populares porque facilitan la ejecución de tareas de uso común en las aplicaciones como crear un proyecto, salvar el proyecto, añadir archivos, correr o depurar la aplicación.
5. **Cuadro de Herramientas:** Es una ventana acopiable que muestra todos los controles con los cuales se realizarán los diseños de las aplicaciones creadas en cualquiera de los lenguajes de programación incluidos en el IDE de Visual Studio 2015.
6. **Explorador de Soluciones:** Muestra el proyecto en uso y todos sus archivos que lo componen.

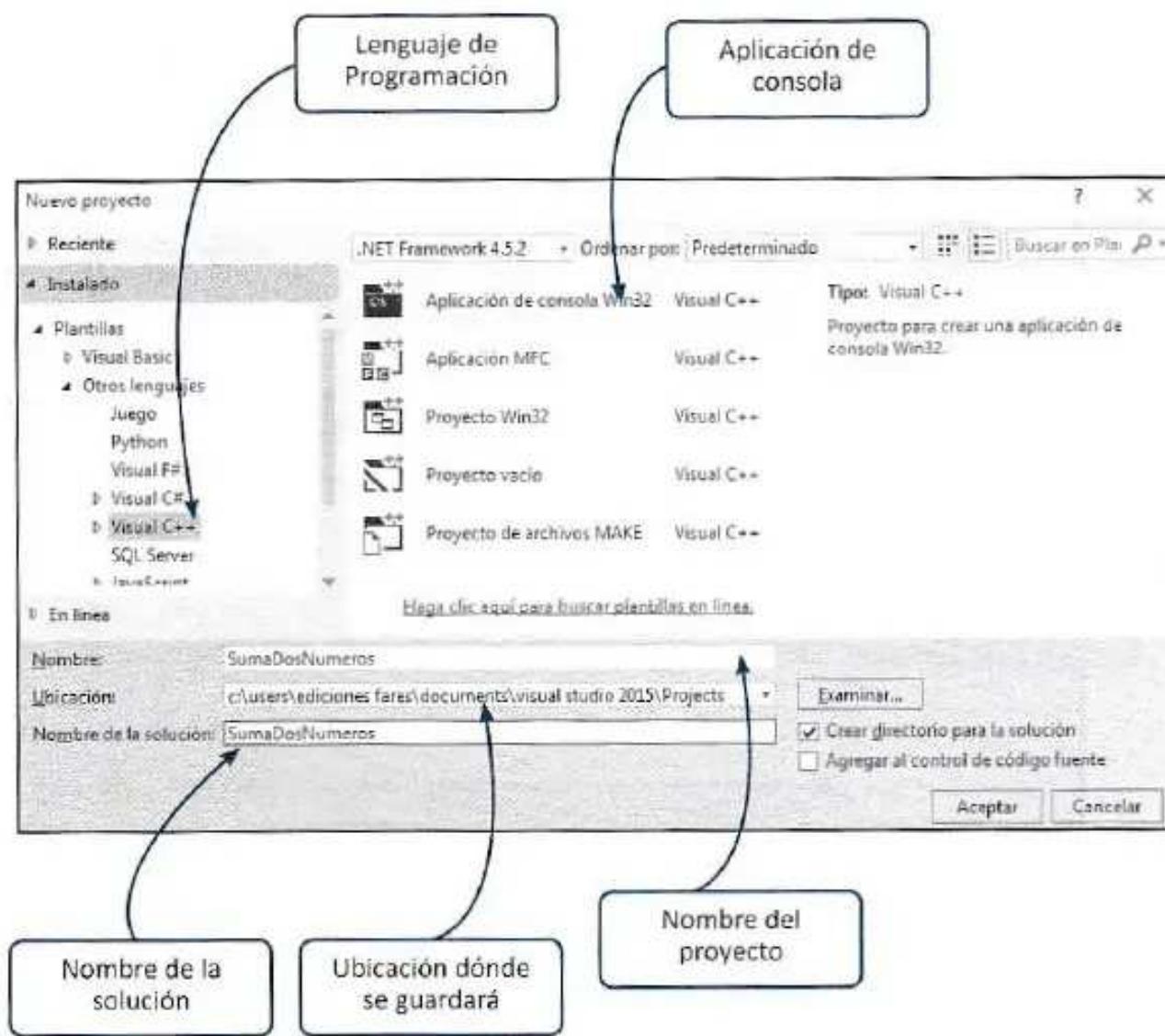
Crear y Guardar un Nuevo Proyecto C++ en Visual Studio Community 2015

Para crear un proyecto realizar las siguientes instrucciones:

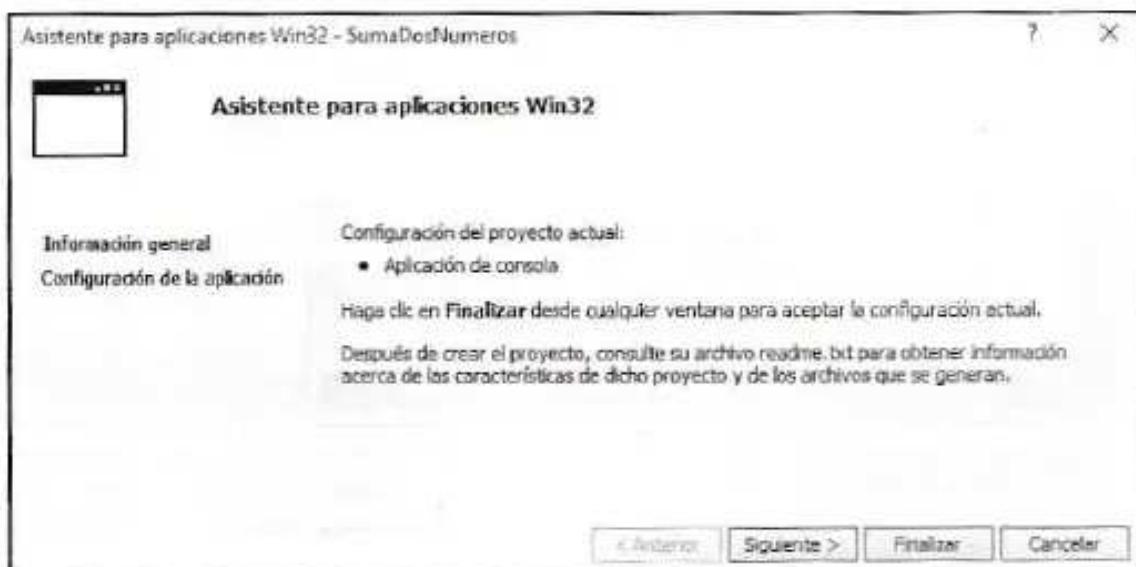
1. Ingresar a Visual Studio Community 2015.
2. Dar click en el menú **Archivo** y a continuación en **Nuevo proyecto** o presionar la combinación de teclas **CTRL + N**.



3. En la Ventana **Nuevo proyecto**, seleccionar la plantilla del lenguaje con el que desea programar (**Visual C++**), el tipo de aplicación que desea crear (**Aplicación de consola Win 32**), seleccionar la ubicación donde desea guardarla (**documents\visual studio 2015\Projects**), dar el nombre de proyecto (**SumaDosNumeros**), dar el nombre de la solución (**SumaDosNumeros**) y a continuación dar click sobre el botón **Aceptar**.



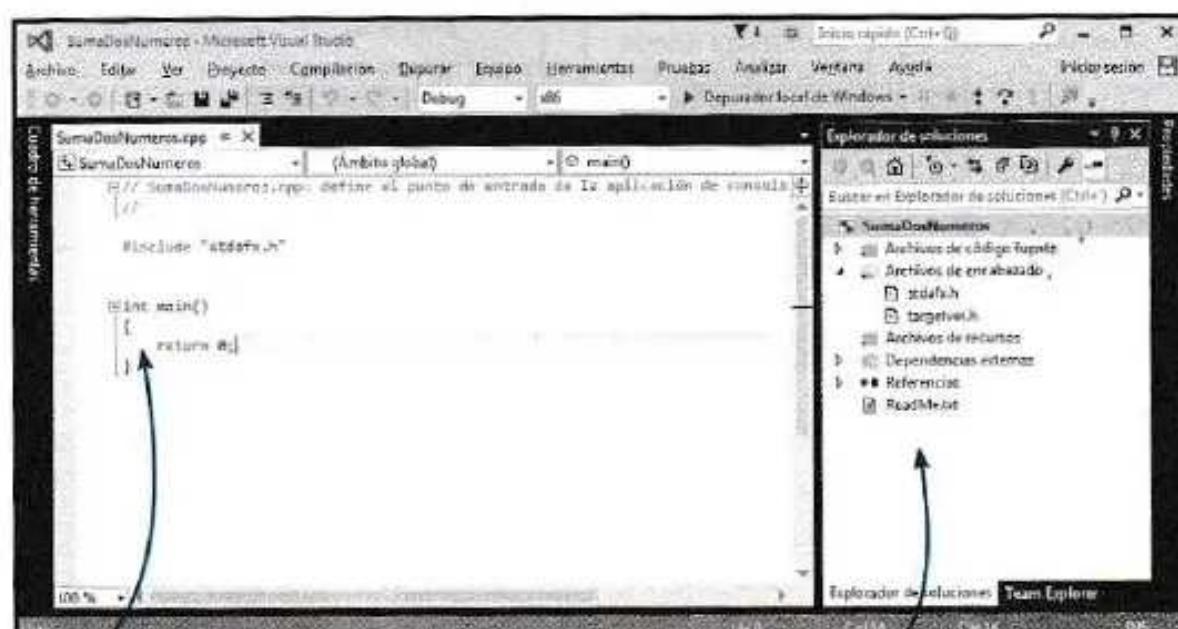
4. Dar click sobre el botón **Siguiente**.



5. Seleccionar el tipo de aplicación con que desea trabajar (**Aplicación de consola**) y dar click en el botón **Finalizar**.

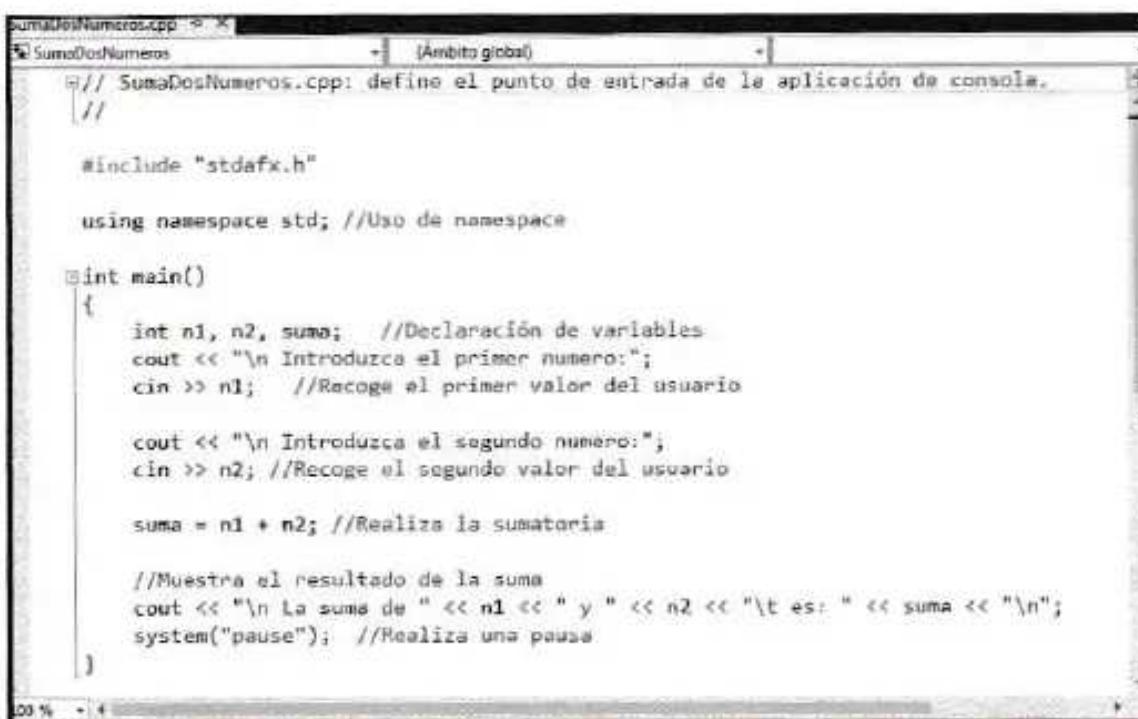


6. Se mostrará la ventana dónde escribiremos el código de nuestra aplicación.



Ejemplos:

1. Crear un programa que realice la suma de dos números y muestre en pantalla su resultado. Para este ejemplo utilizaremos el proyecto (SumaDosNumeros), el programa quedará como la siguiente pantalla:



```

sumaDosNumeros.cpp > X
SumaDosNumeros [Atributo global]
// SumaDosNumeros.cpp: define el punto de entrada de la aplicación de consola.
//
#include "stdafx.h"

using namespace std; //Uso de namespace

int main()
{
    int n1, n2, suma; //Declaración de variables
    cout << "\n Introduzca el primer numero:"; 
    cin >> n1; //Recoge el primer valor del usuario

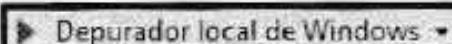
    cout << "\n Introduzca el segundo numero:"; 
    cin >> n2; //Recoge el segundo valor del usuario

    suma = n1 + n2; //Realiza la sumatoria

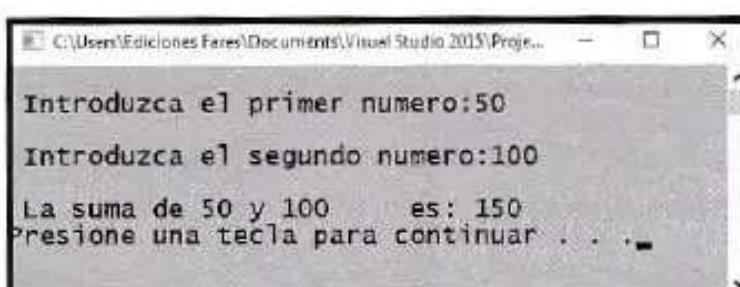
    //Muestra el resultado de la suma
    cout << "\n La suma de " << n1 << " y " << n2 << "\t es: " << suma << "\n";
    system("pause"); //Realiza una pausa
}

```

Para ejecutar el programa dar click en **Depurador local de Windows**.



Se mostrará la siguiente pantalla, dónde el usuario puede introducir los datos:



#include "stdafx.h", incluye el archivo precompilado `stdafx.h`

#include <iostream>, incluye la librería `iostream` que nos permite el control de entrada y salida de datos. La inclusión de esta librería también se puede hacer en el archivo `stdafx.h` y de esa manera dentro del código del programa solamente se llamaría al archivo `stdafx.h`, este archivo quedaría de la siguiente manera:

```

SumaDosNumeros.cpp
SumaDosNumeros.cpp (Ámbito global)
// stddef.h: archivo de inclusión de los archivos
// o archivos de inclusión específicos de un proyecto
// pero nombre vez modificados
//

#pragma once

#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream> //Se utiliza la librería iostream

```

using namespace std;, pone en uso el espacio de nombre std; de esa manera no se tiene que utilizar std en la instrucción cout o cin.

int main(), es la función principal del programa.

{}, las llaves indican el inicio y fin del bloque de sentencias.

int n1, n2, suma;, se declaran las variables n1, n2 y suma de tipo entero.

cout << "\n Introduzca el primer numero:";, escribe en pantalla **Introduzca el primer numero.**

cin >> n1;, permite la introducción de datos a través del teclado, este dato lo tomará la variable n1.

cout << "\n Introduzca el segundo numero:";, \n, hace que el texto **Introduzca el segundo numero,** se muestra en la línea siguiente de dónde se introdujo el primer número.

cin >> n2;, permite la introducción de datos a través del teclado, este dato lo tomará la variable n2.

suma = n1 + n2;, asigna a la variable suma, la suma del número que esté en la variable n1 más el valor que esté en la variable n2.

cout << "\n La suma de " << n1 << " y " << n2 << "\t es: " << suma << "\n";

- ➊ La instrucción “\n La suma de ” muestra en la línea siguiente después de introducir el segundo valor el mensaje **“La suma de”**.
- ➋ La instrucción **<< n1** imprime el primer valor introducido por el usuario.
- ➌ La instrucción **<< “y”** imprime en pantalla la letra y.
- ➍ La instrucción **<< n2** imprime el segundo valor introducido por el usuario.
- ➎ La instrucción **<< “\t es:”** realiza una tabulación horizontal e imprime la palabra es.
- ➏ La instrucción **<< suma** muestra el valor que contiene la variable suma.
- ➐ La instrucción **<< “\n”** ubica el cursor al inicio de la siguiente línea.
- ➑ La instrucción **system("pause");**, realiza una pausa y muestra el siguiente mensaje: **Presione una tecla para continuar....**

2. Crear una aplicación que solicite como dato de entrada el nombre y devuelva como resultado, el

saludo cordial seguido del nombre introducido, el código sería el siguiente:

```

stdafx.h    SolicitarNombre.cpp  * X
SolicitarNombre  (Ámbito global)
// SolicitarNombre.cpp; define el punto de entrada principal
//

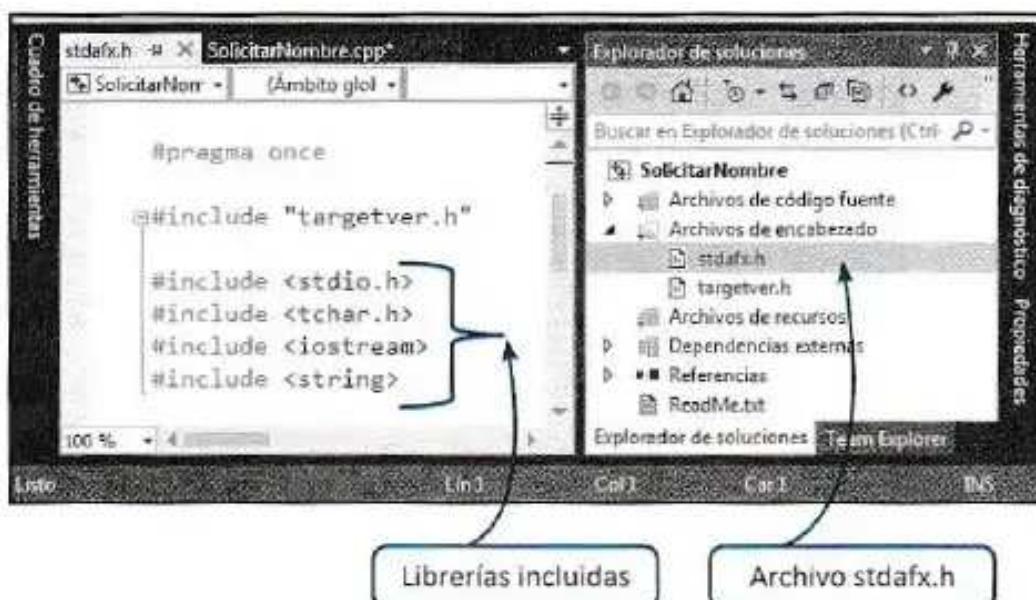
#include "stdafx.h"
using namespace std;

int main()
{
    string s;
    cout << "Escriba su nombre y presione ENTER: ";
    cin >> s;

    cout << "Muy buen dia " << s << "\n";
    system("pause");
}

```

Como puede observar en el código no se incluyeron las librerías, estas se incluyeron en el archivo stdafx.h.



La librería **string**, ayuda a trabajar con texto de longitud indefinida.

string s;, declara la variable de tipo **string**, esta variable recogerá el nombre ingresado por el usuario.

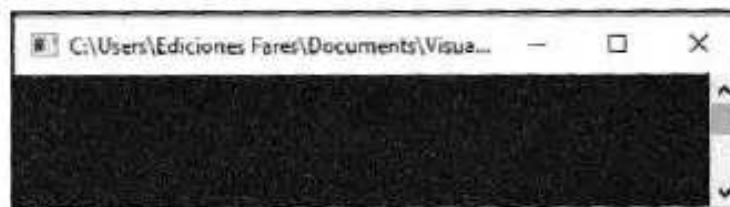
La instrucción **cout << "Escriba su nombre y presione ENTER: "**; imprime en pantalla el mensaje:
Escriba su nombre y presione ENTER.

La instrucción **cin >> s;**, permite que se ingrese el nombre solicitado.

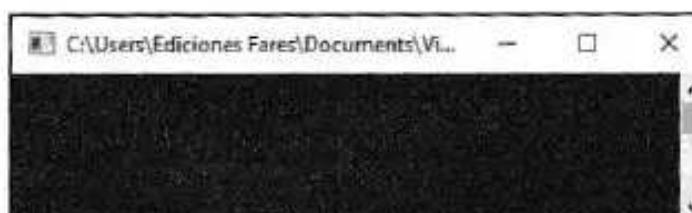
La instrucción `cout << "Muy buen dia " << s << '\n';` imprime en pantalla **Muy buen dia** seguido del nombre que se encuentre en la variable `s`.

La instrucción `system("pause");` detiene la pantalla y muestra el siguiente mensaje: Presione una tecla para continuar....

Al ejecutar el programa se mostrará la siguiente pantalla solicitando el nombre:



Una vez que se ingrese el nombre y se presione **Enter** se mostrará los mensajes: ***Muy buen dia*** ***Fares*** y ***Presione una tecla para continuar....***



3. Crear una aplicación que realice las operaciones de suma, resta, multiplicación y división.

Librerías utilizadas

```
stdafx.h  ▶ X OperacionesAritmeticas.cpp
* OperacionesAritmeticas

#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>

// TODO: mencionar aquí los
```

Código del Programa

```

OperacionesAritmeticas.cpp - (Ámbito global)
// OperacionesAritmeticas.cpp: define el punto de entrada de la aplicación de consola.
//

#include <iostream>
using namespace std;

int main()
{
    cout << "Operaciones Aritmeticas Basicas" << endl;
    int n1, n2, res;
    cout << endl;
    cout << "Ingrese el primer numero: " << endl;
    cin >> n1;
    cout << "Ingrese el segundo numero: " << endl;
    cin >> n2;
    res = n1 + n2;
    cout << "La suma de " << n1 << " + " << n2 << " es: " << res << endl;
    res = n1 - n2;
    cout << endl << "La resta de " << n1 << " - " << n2 << " es: " << res << endl;
    res = n1 / n2;
    cout << endl << "La division de " << n1 << " / " << n2 << " es: " << res << endl;
    res = n1 * n2;
    cout << endl << "El producto de " << n1 << " X " << n2 << " es: " << res << endl;
    cout << endl;
    system("pause");
}

```

Resultado

```

C:\Users\Ediciones Fares\documents\visu...
La suma de 40 + 10 es: 50
La resta de 40 - 10 es: 30
La division de 40 / 10 es: 4
El producto de 40 X 10 es: 400
Presione una tecla para continuar . .

```

Sentencias o Estructuras de Control en C++

Controlan el flujo de control de un programa, permitiendo elegir qué número de veces se ejecuta una determinada instrucción o cuando debe ejecutarse.

Existen dos tipos de estructuras de control:

- Alternativas o de Selección (Decisión)
- Repetitivas o de Iteración

Sentencias Alternativas o de Selección

Ayudan a controlar si una sentencia o bloque de sentencias se ejecutan, en función del cumplimiento o no de una condición o expresión lógica.

Instrucción If

Permite que se ejecuten unas u otras sentencias dependiendo del valor que toma la condición.

Sintaxis de la instrucción If

Alternativa Simple	Explicación
<pre>if (condición) instrucción1;</pre>	Si la condición da como resultado verdadero, se ejecuta la instrucción o sentencia (instrucción1) que sigue a la sentencia if, si la condición da como resultado falso, se ejecutan las sentencias que siguen después de instrucción1.
<pre>if (condición) { instrucción 1; instrucción 2; instrucción 3; }</pre>	Si la condición da como resultado verdadero, se ejecuta el bloque de instrucciones o sentencias (instrucción1, instrucción2, instrucción3) que se encuentran encerradas entre llaves después de la sentencia if, si la condición da como resultado falso, se ejecutan las sentencias que siguen después del bloque de instrucciones.

Alternativa Doble	Explicación
<pre>if (condición) instrucción1; else instrucción2;</pre>	Si la condición da como resultado verdadero, se ejecuta la instrucción1, si la condición da como resultado falso, se ejecuta la instrucción2.
<pre>if (condición) { instrucción 1; instrucción 2; } else { instrucción 3; instrucción 4; }</pre>	Si la condición da como resultado verdadero, se ejecuta instrucción1 e instrucción2, si la condición da como resultado falso, se ejecuta la instrucción3 y la instrucción4. Nota: Recordar que todo bloque de instrucciones se encierra entre llaves.

Ejemplos:

1. Crear un programa que solicite como dato de entrada la edad y devuelva como resultado si es mayor o menor de edad. Cabe recordar, que una persona es mayor de edad cuando tiene 18 o más años.

Librerías incluidas en el archivo stdafx.h

```

stdafx.h  • X
MayoOrNo_de_Edad  (Ámbito global)
// stdafx.h: archivo de inclusión
// o archivos de inclusión específicos
// pero rara vez modificados
//
#pragma once
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>

```

Código del Programa

```

#include "stdafx.h"

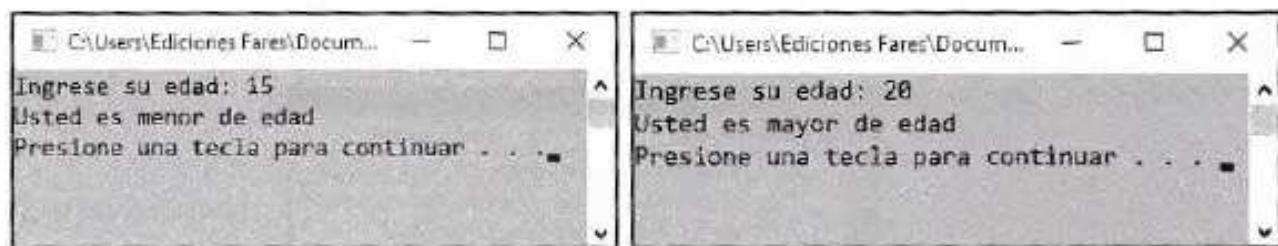
using namespace std;

int main()
{
    int edad;
    cout << "Ingrese su edad: ";
    cin >> edad;
    if (edad >= 18)
    {
        cout << "Usted es mayor de edad" << endl;
    }
    else
    {
        cout << "Usted es menor de edad" << endl;
    }
    system("pause");
}

```

Explicación del Programa

Si la instrucción `if (edad >= 18)` da como resultado verdadero, se ejecuta la instrucción `cout << "Usted es mayor de edad" << endl;` y a continuación se ejecuta la instrucción `system("pause");`; pero si da como resultado falso se ejecuta la instrucción `cout << "Usted es menor de edad" << endl;` y a continuación se ejecuta `system("pause");`.



2. Crear un programa que determine el descuento de 50% en la compra de un tiquet de entrada al estadio, para todas aquellas personas que tengan una edad mayor o igual a 60, el valor del tiquet es de 250 Lps.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
#include "stdafx.h"
using namespace std;

int main()
{
    int edad;
    double desc;
    double valor;
    cout << "Ingrese la edad del cliente: ";
    cin >> edad;
    if (edad < 60)
    {
        desc = 0;
        valor = 250 - desc;
    }
    else
    {
        desc = 250 * 0.5;
        valor = 250 - desc;
    }
    cout << "El valor de su tiquet es: Lps. " << valor << endl;
    system("pause>null");
}
```

Explicación del Programa

Si la instrucción `if (edad < 60)` devuelve como resultado verdadero se realiza (`desc = 0; y valor = 250 - desc;`) y a continuación muestra el mensaje `El valor de su tiquet es: Lps.` y el valor que tenga la variable `valor`; pero si da como resultado falso realiza el descuento para las personas con una edad mayor de 60 (`desc = 250 * 0.5; y valor = 250 - desc;`) y a continuación muestra el mensaje `El valor de su tiquet es: Lps.` y el valor que tenga la variable `valor`. La instrucción `>null` incluida en `system("pause>null");`, hace que no se muestre el mensaje `Presione una tecla para continuar...`.

```
c:\users\ediciones fares\docu... - X
Ingrese la edad del cliente: 20
El valor de su tiquet es: Lps. 250
```



```
c:\users\ediciones fares\doc... - X
Ingrese la edad del cliente: 71
El valor de su tiquet es: Lps. 125
```

- Crear un programa que pida como dato de entrada un número entero y devuelva como resultado un mensaje diciendo si el número ingresado es par o impar.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

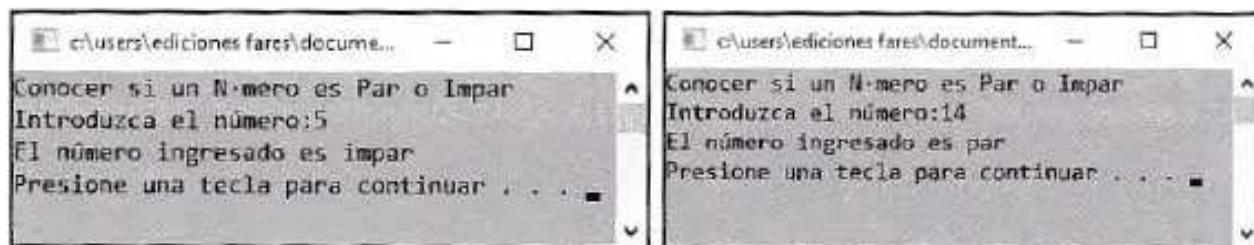
Código del Programa

```
#include "stdafx.h"
using namespace std;

int main()
{
    int num;
    cout << "Conocer si un Número es Par o Impar" << endl;
    cout << "Introduzca el numero:";
    cin >> num;
    if ((num % 2) == 0)
        cout << "El n\xamero ingresado es par" << endl;
    else
        cout << "El n\xamero ingresado es impar" << endl;
    system("pause");
}
```

Explicación del Programa

Si la instrucción if ((num % 2) == 0) devuelve como resultado verdadero se muestra el mensaje El número ingresado es par y a continuación se ejecuta system("pause"); que muestra el mensaje Presione una tecla para continuar...; pero si da como resultado falso se muestra el mensaje El número ingresado es impar y a continuación también se muestra el mensaje Presione una tecla para continuar....



Observar que al ejecutar el programa en la instrucción cout << "Conocer si un Número es Par o Impar" << endl; en Número la ú no se muestra correctamente, sino que se sustituye por otro carácter.

En la instrucción cout << "El n\xamero ingresado es par" << endl; si se muestra la ú con tilde, esta es representada en la instrucción por \xa3.

Los códigos correspondientes a las tildes se muestran en la siguiente tabla:

Carácter	Código Utilizado en C++
á	\xa0
é	\x82
í	\xa1
ó	\xa2
ú	\xa3
Á	\xb5
É	\x90
Í	\xd6
Ó	\xe0
Ú	\xe9
ñ	\xa4
Ñ	\xa5
�	\xa8

4. Determinar el tipo de calificación en relación a la siguiente tabla:

Nota	Calificación
$\geq 0 \text{ y } < 36$	Insuficiente
$\geq 36 \text{ y } < 70$	Necesita Mejorar
$\geq 70 \text{ y } < 81$	Satisfactorio
$\geq 81 \text{ y } \leq 91$	Muy Satisfactorio
$\geq 91 \text{ y } \leq 100$	Avanzado
Caso contrario	Valor fuera de rango

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
#include <string>
```

Código del Programa

```
//Calificación de Notas
//Elaborado por: Ediciones Fares
#include "stdafx.h"
using namespace std;

int main()
{
    string calificacion;
    int nota;
    cout << "Calificar Notas \n";
    cout << "\n Introduzca la nota: ";
    cin >> nota;
    if (nota >= 0 && nota < 36)
    {
        calificacion = "Insuficiente";
    }
    else if (nota >= 36 && nota < 70)
    {
        calificacion = "Necesita Mejorar";
    }
    else if (nota >= 70 && nota < 81)
```

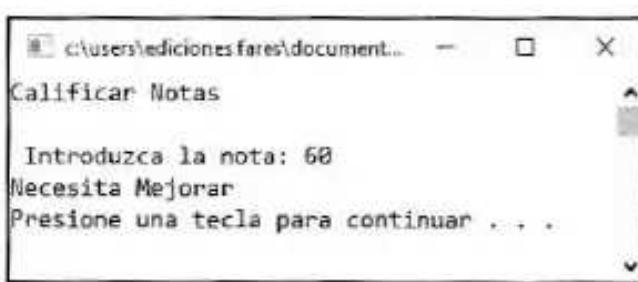
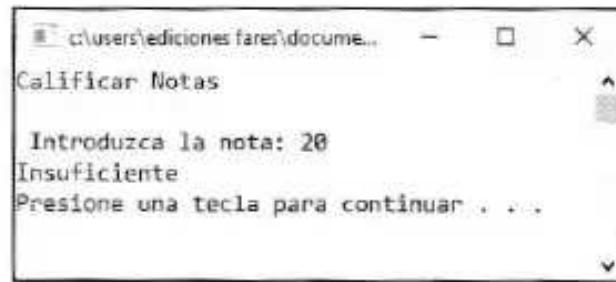
```

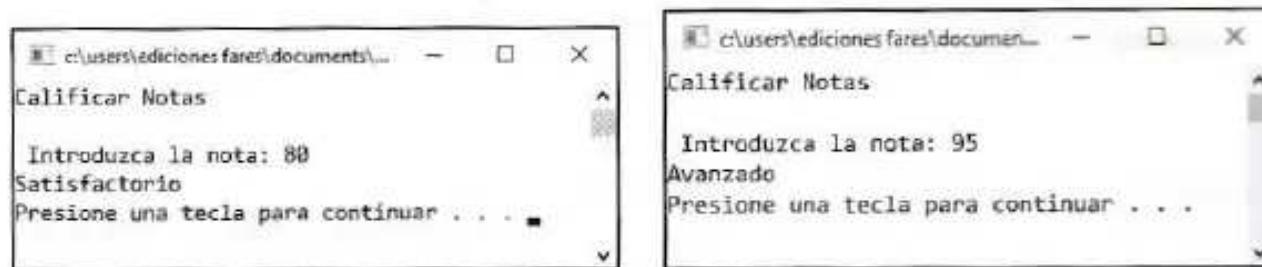
    {
        calificacion = "Satisfactorio";
    }
    else if (nota >= 81 && nota < 91)
    {
        calificacion = "Muy Satisfactorio";
    }
    else if (nota >= 91 && nota < 100)
    {
        calificacion = "Avanzado";
    }
    else
    {
        calificacion = "Valor fuera de rango";
    }
    cout << calificacion << "\n";
    system("pause");
    return 0;
}

```

Explicación del Programa

Si la instrucción `if (nota >= 0 && nota < 36)` devuelve como resultado verdadero se asigna el mensaje **Insuficiente** a la variable `calificacion`, pero si da como resultado falso entonces se evalúa `if (nota >= 36 && nota < 70)`, y si este da como resultado verdadero se le asigna el mensaje **Necesita Mejorar** a la variable `calificacion`, pero si da como resultado falso entonces se evalúa `if (nota >= 70 && nota < 81)`, si este da como resultado verdadero se le asigna el mensaje **Satisfactorio** a la variable `calificacion`, pero si da como resultado falso entonces se evalúa `if (nota >= 81 && nota < 91)`, si este da como resultado verdadero se le asigna el mensaje **Muy Satisfactorio** a la variable `calificacion`, pero si da como resultado falso entonces se evalúa `if (nota >= 91 && nota < 100)` si este da como resultado verdadero se le asigna el mensaje **Avanzado** a la variable `calificacion`; pero si todas las evaluaciones dan como resultado falso se le asigna el mensaje **Valor fuera de rango** a la variable `calificacion`. Una vez que se asigna un valor a la variable `calificacion`, este se muestra en pantalla a través de la instrucción `cout << calificacion << "\n"`; y a continuación también se muestra el mensaje **Presione una tecla para continuar...**





Switch/Case

Permite que se ejecute una o varias sentencias dependiendo del valor de una expresión entera. Se utiliza para agilizar la toma de decisiones múltiples y es muy parecido a la utilización de múltiples if...else if.

Esta sentencia se basa en el valor que puede tomar una variable simple o de una expresión simple llamada **selector**, el tipo de datos de este selector será **int o char**.

Sintaxis de la instrucción Switch/Case

Sintaxis	Explicación
<pre>switch(variable) { case valor_1: Sentencias; case valor_2: Sentencias; . . . case valor_n: Sentencias; default: Sentencias; }</pre>	<p>El cuerpo de la instrucción switch consta de una serie de etiquetas case y una etiqueta opcional default.</p> <p>Cuando se usa la sentencia switch el control se transfiere al punto etiquetado con el case cuya expresión constante coincida con el valor de la expresión entera evaluada dentro del switch. A partir de ese punto, todas las sentencias serán ejecutadas hasta el final del switch, es decir hasta llegar al "]". Esto sucede porque las etiquetas sólo marcan los puntos de entrada después de una ruptura de la secuencia de ejecución, pero no marcan los puntos de salida.</p> <p>Si el valor de la variable coincide con el de la expresión de control, el control se transfiere a la instrucción que sigue a esa etiqueta. Si el valor de la variable no coincide con las constantes de las etiquetas case y hay una etiqueta default, el control se transfiere a la etiqueta default. Si el valor de la variable no coincide con las constantes en las etiquetas case y no hay una etiqueta default se abandonará la sentencia switch.</p>

<pre>switch(variable) { case valor_1: Sentencias; Break; case valor_2: Sentencias; Break; . . . case valor_n: Sentencias; default: Sentencias; }</pre>	<p>Sin una instrucción break, se ejecuta cada instrucción de la etiqueta case coincidente hasta el final de la instrucción switch, incluida default. Pero se puede eludir la ejecución secuencial normal usando la sentencia de ruptura break y así ejecutar sólo parte de las sentencias.</p>
--	--

Ejemplos:

1. Crear un programa que permita ingresar una letra y devuelva como resultado un mensaje especificando si es una vocal o no.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// VocalCase.cpp
//Elaborado por: Ediciones Fares
//

#include "stdafx.h"
using namespace std;

int main()
{
```

```

bool vocal;
char letra;
cout << "Conocer si el Caracter Ingresado es una Vocal: ";
cin >> letra;
switch (letra)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        vocal = true;
        break;
    default:
        vocal = false;
}

if (vocal == true)
{
    cout << "El caracter introducido es una vocal\n\n";
}
else
{
    cout << "El caracter introducido no es una vocal\n\n";
}
system("pause");
return 0;
}

```

Explicación del Programa

En este ejemplo, si el valor de entrada en el switch corresponde a una vocal, la variable **vocal** tomará un valor verdadero, en caso contrario, tomará un valor falso. Si por ejemplo, letra contiene el valor 'a', se cumple el primer case, y la ejecución continúa en la siguiente sentencia: `vocal = true;`, ignorando el resto de los case hasta el break, que nos hace abandonar la sentencia switch.

2. Crear un programa que solicite los números del 1 al 9 y devuelva como resultado el mismo número, pero en letras.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// NumeroALetras.cpp:
//convertir números entre el 1 y el 9 a letras
//Elaborado por: Ediciones Fares

#include "stdafx.h"
using namespace std;

int main()
{
    int n1;
    cout << " Convertir Números a Letras\n\n";
    cout << " Ingrese un número: ";
    cin >> n1;
    switch (n1)
    {
        case 1:
            cout << "\n\n Uno";
            break;
        case 2:
            cout << "\n\n Dos";
            break;
        case 3:
            cout << "\n\n Tres";
            break;
        case 4:
            cout << "\n\n Cuatro";
            break;
        case 5:
            cout << "\n\n Cinco";
            break;
    }
}
```

```

case 6:
    cout << "\n\n Seis";
    break;
case 7:
    cout << "\n\n Siete";
    break;
case 8:
    cout << "\n\n Ocho";
    break;
case 9:
    cout << "\n\n Nueve";
    break;
default:
    cout << "\n\n Valor fuera de rango";
    break;
}

cout << endl << endl;
system("pause");
return 0;
}

```

Explicación del Programa

En este ejemplo, si el valor de entrada en el switch corresponde a un número entero entre 1 y 9, se muestra este mismo pero en letras, en caso contrario, se muestra el mensaje **Valor fuera de rango**. Si por ejemplo, **n1** contiene el valor 1, se cumple el primer case; por lo tanto se muestra el mensaje **Uno** y se abandona la sentencia switch.

- Crear un programa que muestre los precios de la camiseta seleccionada, debe mostrar un menú con los nombres del equipo que representa la camiseta. Tomar los datos de la siguiente tabla:

Precios de Camisetas de Equipos de Fútbol		
Real Madrid	Lps.	3800.00
Barcelona	Lps.	2000.00
Atlético de Madrid	Lps.	1800.00
Manchester United	Lps.	1500.00

Librerías incluidas en el archivo stdafx.h

```

#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>

```

Código del Programa

```
// MenuCamisetasEquipos.cpp:  
//Programa de Menú  
//Elaborado por: Ediciones Fares  
  
#include "stdafx.h"  
using namespace std;  
  
int main()  
{  
    int equipo;  
    cout << "*****" << endl;  
    cout << "\t Precios de Camisetas de Equipos de Futbol" << endl;  
    cout << "\t\t 1.- Real Madrid" << endl;  
    cout << "\t\t 2.- Barcelona" << endl;  
    cout << "\t\t 3.- Atl\x82tico Madrid" << endl;  
    cout << "\t\t 4.- Manchester United" << endl;  
    cout << "\n\n";  
    cout << "\t Seleccione la camiseta: "; cin >> equipo;  
    switch (equipo)  
    {  
        case 1: cout << "\n\t La camiseta del Real Madrid vale 3800 Lempiras" << endl;  
                 break;  
        case 2: cout << "\n\t La camiseta del Barcelona vale 2000 Lempiras" << endl;  
                 break;  
        case 3: cout << "\n\t La camiseta del Atl\x82tico Madrid vale 1800 Lempiras" << endl;  
                 break;  
        case 4: cout << "\n\t La camiseta del Manchester United vale 1500 Lempiras" << endl;  
                 break;  
        default: cout << "\t\t\t Opción no Válida" << endl;  
                  break;  
    }  
    system("pause>null");  
    return 0;  
}
```

Explicación del Programa

En este ejemplo, si el valor de entrada en el switch corresponde a un número entero entre 1 y 4, se muestra el precio de la camiseta seleccionada, en caso contrario, se muestra el mensaje Opción no Válida.

- Crear un programa que solicite el ingreso de dos números enteros, después permita seleccionar si desea sumarlos, restarlos, dividirlos o multiplicarlos y posteriormente, muestre el resultado en pantalla dependiendo de la elección del usuario.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// MenuAritmetico.cpp:
#include "stdafx.h"

using namespace std;

int main()
{
    int opcion, a, b, resultado;

    printf(" Ingrese un numero entero [a]: ");
    scanf_s("%d", &a);

    printf(" Ingrese un numero entero [b]: ");
    scanf_s("%d", &b);

    printf(" Menu\n");
    printf(" 1. Sumar\n");
    printf(" 2. Restar\n");
    printf(" 3. Dividir\n");
    printf(" 4. Multiplicar\n");

    printf(" Elija una operaci\x00f3n: ");
    scanf_s("%d", &opcion);

    switch (opcion)
    {
```

```

        case 1:
            resultado = a + b;
            printf(" %d + %d = %d\n", a, b, resultado);
            break;

        case 2:
            resultado = a - b;
            printf(" %d - %d = %d\n", a, b, resultado);
            break;

        case 3:
            resultado = a / b;
            printf(" %d / %d = %d\n", a, b, resultado);
            break;

        case 4:
            resultado = a * b;
            printf(" %d x %d = %d\n", a, b, resultado);
            break;

        default:
            printf(" Opción no válida\n");
    }
    system("PAUSE");
}

```

Repetitivas o de Iteración

Son aquellas que ejecutan un bloque de sentencias mientras se cumpla una expresión lógica. Este bloque de sentencias que se ejecuta repetidas veces, se denomina **bucle** y a cada ejecución se denomina **iteración**.

While

Permite la ejecución de un bloque de sentencias si se evalúa como verdadera una expresión lógica. La expresión lógica aparece al principio del bloque de sentencias.

Características:

- Repite el cuerpo del ciclo varias veces o ninguna, a esto se le llama de *cero o más iteraciones*.
- El cuerpo del ciclo se repetirá mientras la condición sea verdadera.
- El ciclo terminará cuando la condición sea falsa.

Sintaxis	Explicación
While (Expresión Lógica)	El bloque delimitado por las llaves puede reducirse a una sentencia, y en este caso se suprimen las llaves.
{	
Sentencias;	La expresión lógica debe estar delimitada por paréntesis.
}	
	Cuando el programa llega a una sentencia while, sigue los siguientes pasos: <ul style="list-style-type: none"> • Evalúa la expresión. • Si es falsa, continúa la ejecución tras el bloque de sentencias. • Si es verdadera entra en el bloque de sentencias asociado al while. • Ejecuta el bloque de sentencias, evaluando de nuevo la expresión y actuando en consecuencia. Si la primera evaluación resulta falsa, el bloque de sentencias no se ejecuta nunca. Si la expresión es siempre cierta el bucle es infinito.

Ejemplos:

- Crear un programa que sume todos los números de 1 a 10.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```

// Suma_1_10.cpp:
// Suma todos los números del 1 al 10

#include "stdafx.h"

using namespace std;

int main()
{
    int n1 = 1; //Declaración de la variable n1
    int resul = 0;
    cout << "\tSuma los numeros del 1 al 10" << endl;
    while (n1<=10) //Mientras n1 <= 10
        {//se ejecuta lo que está entre llaves
            cout << "\t\t" << n1 << endl; //Muestra el valor que tiene n1
            resul = resul + n1; //Se suma el valor de n1 a resul
            n1 += 1; //se incrementa en 1 la variable n1
        }
    //Cuando finaliza el conteo muestra la suma total
    cout << "\tSuma total: " << resul << "\n\n";
    system("pause");
    return 0;
}

```

Explicación

Las instrucciones que están entre llaves dentro del while se ejecutan mientras la variable **n1 <= 10**; cuando n1 llega a valer 11, se sale del bucle while y se muestra la suma total y el mensaje **Presione una tecla para continuar....**

La variable n1 se aumenta en 1 para cada iteración de manera manual en la sentencia **n1 += 1;** //se incrementa en 1 la variable n1.

```
c:\users\ediciones fares\documentos\programas> Suma los numeros del 1 al 10
1
2
3
4
5
6
7
8
9
10
Suma total: 55
Presione una tecla para continuar . . .
```

2. Escribir un programa que solicite la entrada de 10 números y muestre como salida la suma de los mismos.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// SumaNumerosCualesquiero.cpp:
// Suma 10 números introducidos por el usuario
// Elaborado por: Ediciones Fares

#include "stdafx.h"
using namespace std;

int main()
{
    int n1 = 0, n2=0; //Declaración de la variable n1 y n2
    int resul = 0;
    cout << "\tSuma los numeros del 1 al 10\n\n";
    while (n1++ < 10) //Mientras el incremento n1 <= 10
    {
        //se ejecuta lo que está entre llaves
        cout << "\tingrese un n\xadmero: "; cin >> n2; //Muestra el valor que tiene n1
        resul = resul + n2; //Se suma el valor de n2 a resul
    }
}
```

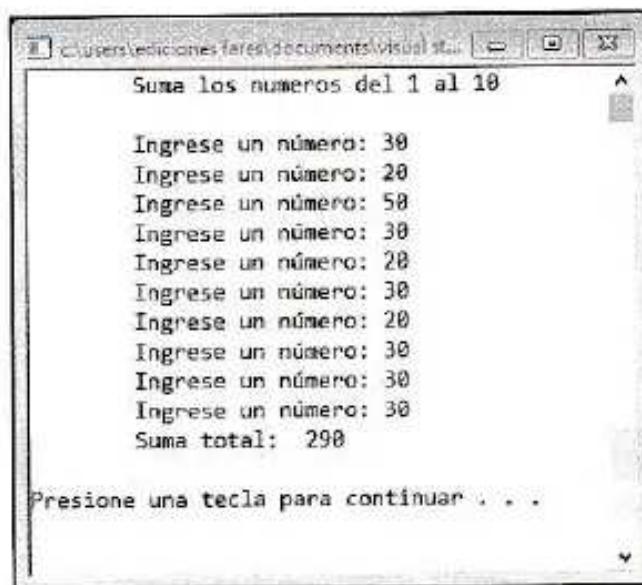
```

    }
    //Cuando finaliza el conteo muestra la suma total
    cout << "\tSuma total: " << resul << "\n\n";
    system("pause");
    return 0;
}

```

Explicación

En este ejemplo, el incremento de la variable **n1** se realiza en la misma instrucción while con la sentencia **n1++**.



3. Crear una aplicación que muestre los números pares entre 1 y 100.

Librerías incluidas en el archivo stdafx.h

```

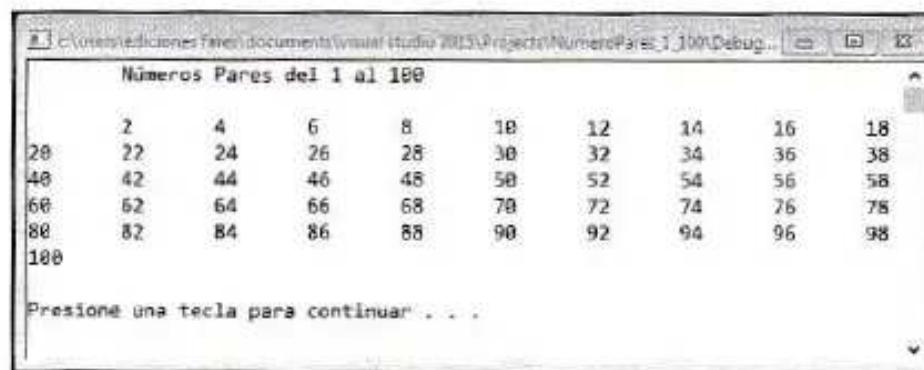
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>

```

Código del Programa

```
// NumeroPares_1_100.cpp:
#include "stdafx.h"
using namespace std;

int main()
{
    int n1 = 0, n2 = 0; //Declaración de la variable n1 y n2
    int resul = 0;
    cout << "\tN\x03meros Pares del 1 al 100\n\n";
    while (n1++ < 100) //Mientras el incremento n1 < 100
    { //se ejecuta lo que está entre llaves
        if (n1 % 2 == 0) //si el residuo de dividir el valor en n1
        { // por 2 es igual a 0, se muestra el número que tiene n1.
            cout << "\t" << n1; //Muestra el valor que tiene n1
        }
    }
    cout << "\n\n";
    system("pause");
    return 0;
}
```



Do....While

Permite la ejecución de un bloque de sentencias si se evalúa como verdadera una expresión lógica. A diferencia de while, do...while primero ejecuta el cuerpo del ciclo al menos una vez y luego evaluará el control de detención.

Sintaxis	Explicación
<pre>do { Sentencias; } while (condición);</pre>	<p>El bloque delimitado por las llaves puede reducirse a una sentencia, y en este caso se suprimen las llaves.</p> <p>La condición lógica debe estar delimitada por paréntesis.</p> <p>Cuando el programa llega a una sentencia while, seguir los siguientes pasos:</p> <ul style="list-style-type: none"> • Evalúa la expresión. • Si es falsa, continúa la ejecución tras el bloque de sentencias. • Si es verdadera ejecuta en el bloque de sentencias entre do y while. • Ejecuta el bloque de sentencias al menos una vez antes de ser evaluada la condición. <p>Si la primera evaluación resulta falsa, el bloque de sentencias se ejecuta al menos una vez.</p> <p>Si la expresión es siempre cierta el bucle es infinito.</p>

Ejemplos:

1. Crear un programa que sume todos los números de 1 a 10.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```

// suma1_10DoWhile.cpp:
//

#include "stdafx.h"

using namespace std;

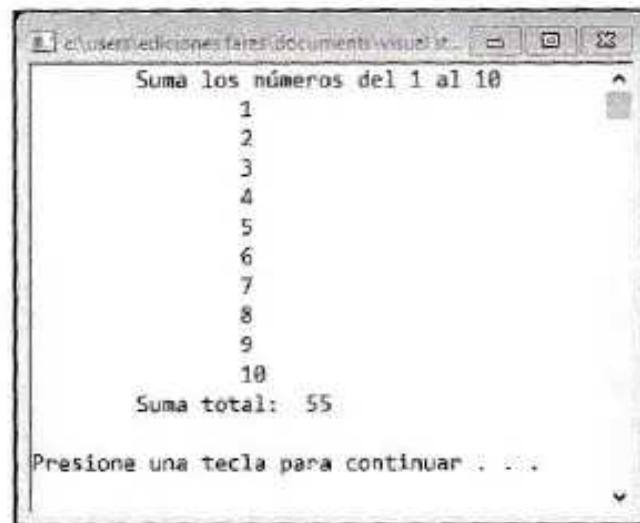
int main()
{
    int n1 = 1; //Declaración de la variable n1
    int resul = 0;
    cout << "\tSuma los números del 1 al 10" << endl;
    do
    {
        //se ejecuta lo que está entre llaves
        cout << "\t\t" << n1 << endl; //Muestra el valor que tiene n1
        resul = resul + n1; //Se suma el valor de n1 a resul
        n1 += 1; //se incrementa en 1 la variable n1
    }
    while (n1 <= 10); //Evalúa si n1 <= 10
    //Cuando n1 = 11 muestra la suma total
    cout << "\tSuma total: " << resul << "\n\n";
    system("pause");
    return 0;
}

```

Explicación

Las instrucciones que están entre llaves dentro de do...while se ejecutan mientras la variable `n1 <= 10`; cuando `n1` llega a valer 11, se sale del bucle do...while y se muestra la suma total y el mensaje [Presione una tecla para continuar...](#)

La variable `n1` se aumenta en 1 para cada iteración de manera manual en la sentencia `n1 += 1; //se incrementa en 1 la variable n1`.



2. Escribir un programa que solicite la entrada de varios números y muestre como salida la suma de los mismos. El programa debe finalizar cuando el usuario ingrese un cero (0).

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// SumaNumero_DoWhile.cpp:
// Suma infinitos números

#include "stdafx.h"
using namespace std;

int main()
{
    int n1 = 1, n2 = 0; //Declaración de la variable n1 y n2
    int resul = 0;
    cout << "\tSuma Infinitos N\xfa3meros\n\n";
    do
        {//se ejecuta lo que est\xfa entre llaves
            cout << "\tingrese el valor " << n1 << "\t"; cin >> n2; //Muestra el valor que tiene n1
            resul = resul + n2; //Se suma el valor de n2 a resul
            n1++; //incrementa en 1 la variable n1
        }
}
```

```

}

while (n2 != 0); //Evalúa que n2 sea distinto de cero

//Cuando finaliza el conteo muestra la suma total

cout << "\tSuma total: " << resul << "\n\n";

system("pause");

return 0;

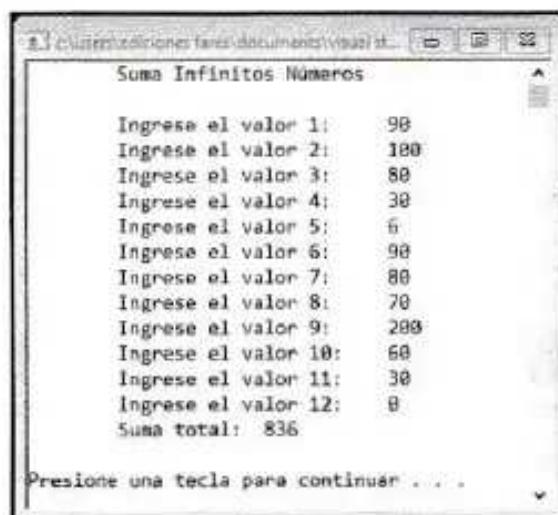
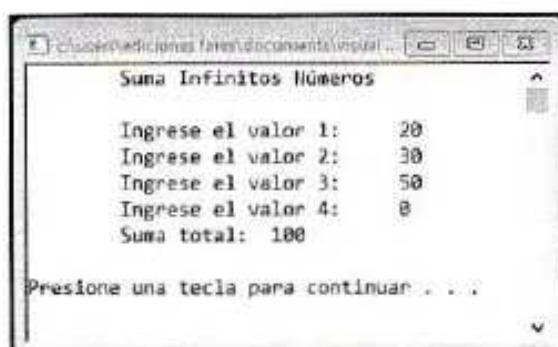
}

```

Explicación

Las instrucciones que están entre llaves dentro de do...while se ejecutan mientras la variable n2 sea distinta de 0; cuando n2 vale 0, se sale del bucle do...while y se muestra la suma total y el mensaje **Presione una tecla para continuar...**

La variable n1 se aumenta en 1 para cada iteración de manera manual en la sentencia n1 += 1; //se incrementa en 1 la variable n1.



- Crear una aplicación que muestre los números pares entre 1 y 100.

Librerías incluidas en el archivo stdafx.h

```

#include "targetver.h"

#include <stdio.h>

#include <tchar.h>

#include <iostream>

```

Código del Programa

```

// Pares1_100_DoWhile.cpp:
// Muestra los números pares entre 1 y 100

#include "stdafx.h"
using namespace std;

int main()
{
    int n1 = 1, n2 = 0; //Declaración de la variable n1 y n2
    int resul = 0;
    cout << "\tN\x03meros Pares del 1 al 100\n\n";
    do
        //se ejecuta lo que está entre llaves
        if (n1 % 2 == 0) //si el residuo de dividir el valor en n1
        { // por 2 es igual a 0, se muestra el número en n1.
            cout << "\t" << n1; //Muestra el valor que tiene n1
        }
    }
    while (n1++ < 100); //Mientras el incremento n1 < 100
    cout << "\n\n";
    system("pause");
    return 0;
}

```

Explicación

Las instrucciones que están entre llaves dentro de do...while se ejecutan mientras la variable n1 sea menor de 100; cuando n2 vale 100, se sale del bucle do...while y el mensaje Presione una tecla para continuar...

Números Pares del 1 al 100								
	2	4	6	8	10	12	14	16
20	22	24	26	28	30	32	34	36
40	42	44	46	48	50	52	54	56
60	62	64	66	68	70	72	74	76
80	82	84	86	88	90	92	94	96
100								98

Presione una tecla para continuar . . .

For

Permite la ejecución de un bloque de sentencias mientras la condición resulte verdadera; en otras palabras, la sentencia es ejecutada repetidamente hasta que la evaluación de la condición sea falsa.

Sintaxis	Explicación
<pre>for ([< inicialización>]; [<condición>] ; [<incremento>]) { <sentencia>; }</pre>	<p>El bloque delimitado por las llaves puede reducirse a una sentencia, y en este caso se suprimen las llaves.</p> <ul style="list-style-type: none"> ● Inicialización: Representa una sentencia de asignación a una variable a la que se le asigna un valor inicial. ● Condición: Representa la condición que finalizará el ciclo cuando esta llegue a ser falsa. ● Incremento: Es una expresión que aumenta en un valor dado al contador o variable representada en la inicialización. <p>Una estructura For, tiene las siguientes características:</p> <ul style="list-style-type: none"> ● Es utilizada dónde se conoce con exactitud el número de iteraciones. ● Es útil cuando sus iteraciones son controladas por un contador. ● Es un ciclo dónde su cuerpo se ejecuta una vez por cada incremento en el contador. ● La inicialización, la condición y el incremento del contador se coloca en la cabecera de la estructura.

Ejemplos:

1. Crear un programa que solicite un número entero y devuelva como resultado la tabla de multiplicar de 1 a 15 para el número ingresado.

Librerías incluidas en el archivo stdafx.h
<pre>#include "targetver.h" #include <stdio.h> #include <tchar.h> #include <iostream></pre>

Código del Programa

```
// Tabla de Multiplicar_For.cpp:  

//  

#include "stdafx.h"  

using namespace std;  

int main()  

{  

    int n; //n Representa el número ingresado por el usuario  

    int contador; //Esta variable lleva el conteo del ciclo  

    int opera; //Almacenará el resultado de multiplicar el número n por el contador  

    cout << "\tTabla de Multiplicar \n\n";  

    cout << "\tIngrese el número: "; cin >> n;  

    for (contador = 0; contador <= 15; contador++)  

    {  

        opera = contador * n; //Realiza la multiplicación  

        cout << "\t" << n << " X " << contador << "=" << opera << "\n";  

    }  

    cout << "\n\n\t";  

    system("pause");  

}
```

Explicación

En la instrucción `for (contador = 0; contador <= 15; contador++)` la variable `contador` se inicializa en 0, el bucle se va a ejecutar mientras la variable `contador` sea `<= 15` y por último, la variable `contador` se incrementa en 1 en cada iteración del ciclo.

The figure consists of two side-by-side screenshots of Microsoft Visual Studio 2015. Both windows have a title bar that reads "Clase 10 Ejercicios de bucles en C++ (Visual Studio 2015)".

Left Window (Multiplication Table for 10):

```
Ingrese el número: 10
10 X 0 = 0
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
10 X 11 = 110
10 X 12 = 120
10 X 13 = 130
10 X 14 = 140
10 X 15 = 150

Presione una tecla para continuar . . .
```

Right Window (Multiplication Table for 9):

```
Ingrese el número: 9
9 X 0 = 0
9 X 1 = 9
9 X 2 = 18
9 X 3 = 27
9 X 4 = 36
9 X 5 = 45
9 X 6 = 54
9 X 7 = 63
9 X 8 = 72
9 X 9 = 81
9 X 10 = 90
9 X 11 = 99
9 X 12 = 108
9 X 13 = 117
9 X 14 = 126
9 X 15 = 135

Presione una tecla para continuar . . .
```

2. Crear un programa que calcule el factorial de un número.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"  
  
#include <stdio.h>  
  
#include <tchar.h>  
  
#include <iostream>
```

Código del Programa

```
// FactorialNumero.cpp  
  
//Devuelve el factorial de un número  
  
//Elaborado por: Ediciones Fares  
  
#include "stdafx.h"  
  
using namespace std;  
  
int main()  
{  
  
    int i, num, fact = 1;  
  
    cout << "Programa que calcula el factorial de un n\xba\x83mero\n";  
  
    cout << "\nEscriba un n\xba\x83mero entero: ";  
    cin >> num;  
  
    for(i = num; i > 1; i--)  
    {  
        fact = fact*i;  
    }  
    cout << "\nEl factorial de " << num << " es " << fact << "\n\n";  
    system("pause");  
    return 0;  
}
```

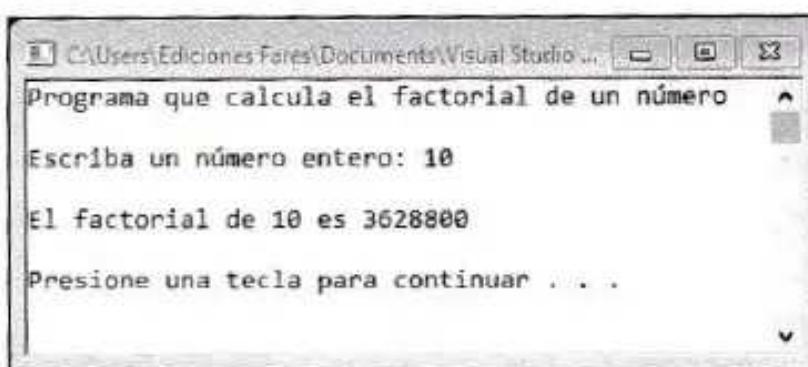
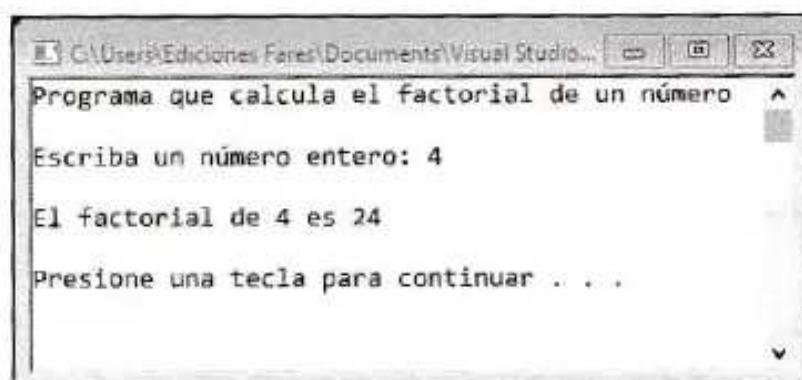
Explicación

En la instrucción `for(i = num; i > 1; i--)` la variable `i` se inicializa con el valor ingresado por el usuario, el bucle se va a ejecutar mientras la variable `i` sea `>1` y por último, la variable `i` realiza un decremento en 1 en cada iteración del ciclo.

La instrucción `fact = fact*i;` realiza la siguiente actividad, dependiendo del valor de `i`; por ejemplo: si se introduce el número 4.

Iteración 1	1	=	1 * 4	4
Iteración 2	4	=	4 * 3	12
Iteración 3	12	=	12 * 2	24
Iteración 4	24	=	24 * 1	24

Al finalizar el resultado que se imprime es 24.



- Crear un programa que indique si un número es primo o no.

Antes de elaborar cualquier programa, lo primero es entender el problema como lo vimos en la sección de los algoritmos para que nos sea más fácil comprender los comandos que utilizaremos al programar. Las preguntas a respondernos en este problema serían:

¿Cómo sé que un número es primo o no?

R.- *Un número es primo cuando se es divisible por sí mismo y por la unidad.*

¿Cómo determino si 5 es un número primo o no?

R.- Divido el número 5 por 1, 2, 3, 4, 5 y si la división es exacta sólo entre él y el 1 entonces es un número primo.

Comprobación:

$$5 \% 1 = 0$$

$$5 \% 2 = 1$$

$$5 \% 3 = 2$$

$$5 \% 4 = 1$$

$$5 \% 5 = 0$$

¿Qué comandos debo utilizar para realizar esta tarea?

R.- Para cualquier número se realiza una misma actividad desde el 1 hasta el número indicado, por lo que es necesario utilizar ciclos, para este ejercicio utilizaremos For.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// NumeroPrimo.cpp:
#include "stdafx.h"

using namespace std;

int main()
{
    int num, i = 1, resto = 0, resto1 = 0;

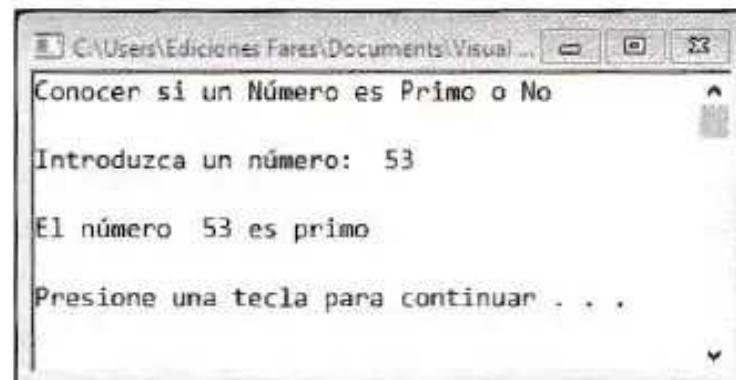
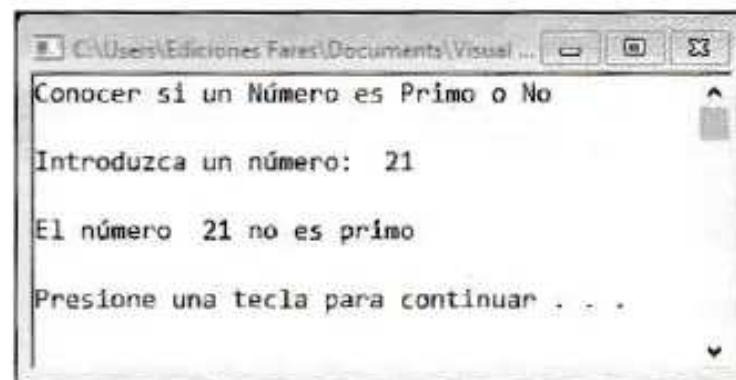
    cout << "Conocer si un Número es Primo o No\n\n";
    cout << "Introduzca un número: "; cin >> num;
    for (int i = 1; i <= num; i++)
    {
        if ((num % i) == 0)
        {
            resto += 1;
        }
    }
}
```

```
    }

    if (resto == 2)
    {
        cout << "\nEl número " << num << " es primo";
    }
    else
    {
        cout << "\nEl número " << num << " no es primo";
    }

    cout << "\n\n";

    system("pause");
    return 0;
}
```

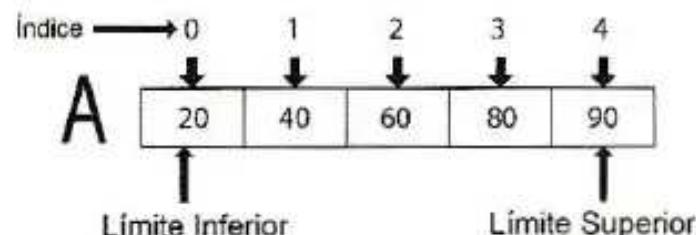


Arreglos

Un Arreglo es un conjunto de datos que se almacenan en memoria de manera contigua bajo un mismo nombre. En otras palabras, podemos decir que una variable de arreglo o vector puede contener al mismo tiempo diferentes valores bajo un mismo nombre.

Uno de los trabajos más comunes en programación es trabajar con listas de valores como: una lista de artículos a comprar en el supermercado o los números de identidad de los trabajadores de una empresa. Las variables de tipo primitivo no pueden manejar listas de valores ya que tienen carácter destructivo; esto implica que, si una variable tiene un valor y se le asigna otro, pierde el primero y mantiene el nuevo. La estructura en sí misma de un arreglo es una colección de objetos a la cual se hace referencia a través del nombre del contenedor y del índice de ubicación. Los datos almacenados en un arreglo se pueden buscar, ordenar, eliminar y hacer referencia a cada uno de los elementos mediante el índice.

Cada dato que se almacena en un arreglo se le llama **elemento** y se hace referencia a él por medio de la posición que ocupa. Los elementos de un arreglo se enumeran consecutivamente, comenzando en cero e incrementando su valor en la unidad hasta el número máximo del arreglo determinado en su declaración.



Cuando se crea una variable de arreglo a cada una de sus casillas se le reconoce por un número al que se le denomina índice. Este valor permite al programador especificar la casilla con la que desea trabajar, usando el siguiente formato:

Nombre_de_arreglo[índice]

En donde **nombre_de_arreglo**, es el nombre de la variable e índice es un valor que indica la casilla con la que se desea trabajar en cada instante. *Ejemplo:*

A[4] → hace referencia al elemento ubicado en el índice cuatro, haciendo referencia al valor 90.

Un arreglo se caracteriza por:

1. Almacenar los elementos del arreglo en posiciones de memoria continua.
2. Tener un único nombre de variable que representa a todos los elementos, y estos a su vez se diferencian por un índice o subíndice.
3. Acceso directo o aleatorio a los elementos individuales del arreglo.

Arreglos Unidimensionales

Es un tipo de datos estructurado que está formado de una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales.

Declaración de un Arreglo

Al declarar un arreglo se elabora igual que cualquier tipo de dato simple, con la diferencia que se debe indicar entre corchetes el tamaño del arreglo.

Su sintaxis es:

Tipodato identificador [Cantelementos]

Donde:

- **Tipodato:** Es el tipo de dato que almacenarán los elementos del arreglo, los elementos de un arreglo sólo pueden almacenar un tipo de dato.
- **Identificador:** Es el nombre que se le dará al arreglo.
- **Cantelementos:** Es un número entero que determina la cantidad de elementos que tendrá el arreglo.

Ejemplo:

Int Arreglo1[4]	Arreglo[0] Arreglo[1] Arreglo[2] Arreglo[3]
-----------------	--

Un elemento del arreglo se accede indexando el arreglo por medio de un número del elemento. En C++ todos los arreglos empiezan en 0, esto quiere decir que, si se quiere acceder al primer elemento del arreglo, se utilizaría el índice 0. Para indexar un arreglo se especifica el índice del elemento que interesa dentro de un corchete.

Ejemplos:

1. Crear un programa que cargue la tabla de multiplicar en un arreglo (en una variable se almacenará el producto y en otra los números consecutivos del 0 al 15) para cualquier número ingresado por el usuario. El número ingresado por el usuario debe multiplicarse por los números consecutivos del 0 al 15.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"  
#include <stdio.h>  
#include <tchar.h>  
#include <iostream>
```

Código del Programa

```
// TablaMultiplicarArreglo.cpp:  
// Elaborado por: Ediciones Fares  
#include "stdafx.h"  
  
using namespace std;  
  
int main()  
{  
    int producto[16], multiplicando[16];  
  
    int multiplicador = 0, x=0;  
  
    cout << "Ingrese un n\xamero: "; cin >> multiplicador;  
  
    if (multiplicador != 0)  
    {  
        for (x = 0; x <= 15; x++)  
        {  
            producto[x]=x * multiplicador;  
  
            multiplicando[x] = x;  
        }  
  
        for (x = 0; x <= 15; x++)  
        {  
            cout << "\n" << multiplicador << " * " << multiplicando[x] << " = " <<  
            producto[x];  
        }  
    }  
  
    cout << "\n\n";  
    system("pause");  
}
```

Explicación

La variable producto[16] se declaró con un espacio de 16 ya que almacenará los números del 0 al 15 y si los contamos nos daremos cuenta que hay 16 números.

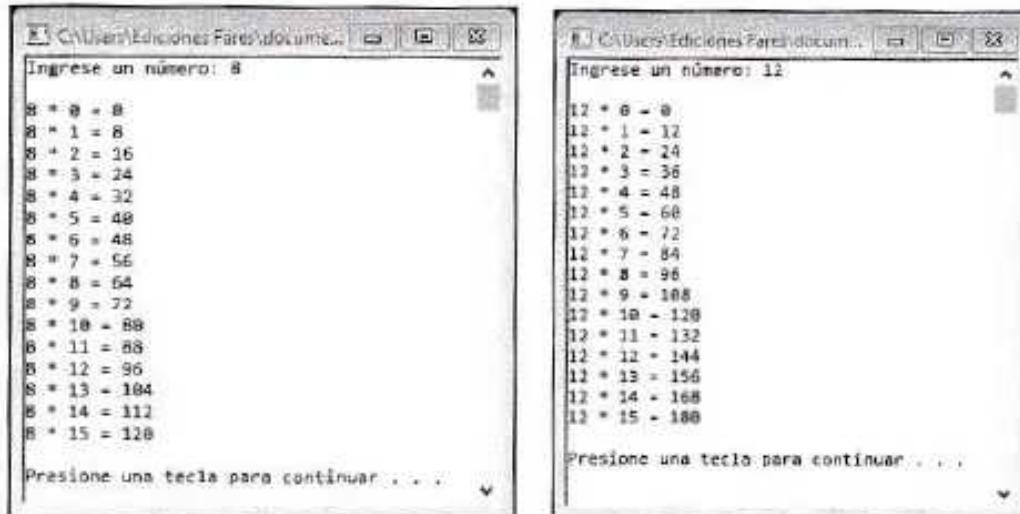
Con la instrucción if (multiplicador != 0) se verifica el número ingresado, si este es igual a cero sale del If y se muestra el mensaje Presione una tecla para continuar..., pero si el número ingresado es distinto de cero se realiza el cálculo.

```
for (x = 0; x <= 15; x++)
{
    producto[x]=x * multiplicador;
    multiplicando[x] = x;
}
```

En el bloque de texto anterior se creó un ciclo que se ejecutará desde el cero hasta el 15. La sentencia `producto[x]=x * multiplicador;` indica que se almacenará en cada posición del arreglo el producto obtenido de multiplicar el valor de la variable `x` por el valor de la variable `multiplicador`. La instrucción `multiplicando[x] = x;` indica que se almacenará en cada posición el valor de la variable `x`.

```
for (x = 0; x <= 15; x++)
{
    cout << "\n" << multiplicador << " * " << multiplicando[x] << " = " << producto[x];
}
```

En la porción de texto anterior, se creó un ciclo que se ejecutará desde el cero hasta el 15. En la sentencia `cout` se imprimirá el valor de la variable `multiplicador`, el valor que se encuentre en la posición del valor que tenga la variable `x` en ese momento del arreglo `multiplicando`; de igual forma el del arreglo `producto`.



2. Crear un programa que almacene en un array los números del 0 al 20 y los muestre en pantalla en orden descendente.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"  
#include <stdio.h>  
#include <tchar.h>  
#include <iostream>
```

Código del Programa

```
// ArregloDe20Numeros.cpp:  
//Elaborado por: Ediciones Fares  
  
#include "stdafx.h"  
using namespace std;  
  
int main()  
{  
    int x,p, tabla[19];  
    x = 0;  
    p = 19;  
    for (x = 0; x <= 19; x++)  
    {  
        tabla[x] = x + 1;  
    }  
    for (p = 19; p!=0; p--)  
    {  
        cout << "\n" << tabla[p];  
    }  
    cout << "\n\n";  
    system("pause");  
}
```



Estrategia de Aprendizaje # 1

Instrucciones: Realizar las actividades que se le muestran a continuación:

Crear un programa que realice la planilla de una empresa.

1. El programa debe solicitar el nombre y salario mensual del empleado.
2. El programa debe solicitar el número de días faltados.
3. El programa debe realizar el cálculo de las deducciones.
 - a. Salario de los días faltados.
 - b. IHSS (Si el salario >7000, salario * 3.5%, 7000 * 3.5%).
4. El programa debe realizar la suma de las deducciones (diasfaltados + IHSS).
5. El programa debe realizar mostrar el salario neto (salario – totaldeducciones).

Introducción a la Programación Orientada A Objetos en C++ (POO)

Es un tipo de programación que usa objetos para el diseño de aplicaciones y programas informáticos.

La programación orientada a objetos busca amoldarse al modo de pensar del hombre y no al de la máquina. Esto es posible gracias a la forma con la que se manejan las abstracciones que representan las entidades del dominio del problema y a propiedades como la jerarquía o el encapsulamiento.

Con la **POO** es necesario aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de objetos, propiedades y métodos; pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo, vamos a pensar en un automóvil para tratar de modelizarlo en un esquema de POO. Díjamos que el automóvil es el elemento principal que tiene una serie de características como ser: el color, modelo o marca. Además, tiene una serie de funcionalidades asociadas, como ser: ponerse en marcha, parar o aparcar, etc.

Elementos de la Programación Orientada Objetos

Objeto

Es la entidad básica en tiempo de ejecución en un sistema orientado a objetos; este puede representar una persona, cuenta bancaria, tabla de alumnos o cualquier elemento que el sistema tenga que manipular.

Características de los Objetos

Por ser entes que pueden ser manipulados en el mundo real, los objetos tienen características que los distinguen. En este libro se exponen tres grandes elementos:

- Atributos:** Son características que definen al objeto; también son conocidas como *propiedades* y proporcionan información acerca del ente. Por ejemplo, una pizarra tiene varios atributos: su tamaño, color, forma, dimensiones, etc. No deben confundirse los atributos con sus valores. Muchas personas piensan que un atributo de una computadora es que es negra., sin embargo, la propiedad es el color y el valor de esta propiedad, es negro.
- Métodos:** Son operaciones que un objeto puede llevar a cabo para transformar el entorno en que se encuentran o para comunicarse con otros objetos. Un automóvil posee un método de arranque, el cual le permite ponerse en marcha, así como posee un método de frenado que le permite detenerse. Un teléfono celular posee un método de emisión de señal, que le permite comunicarse con otros aparatos del mismo tipo.
- Eventos:** Es una acción llevada a cabo por un ente externo que modifica el estado del objeto. *Por ejemplo:* Una persona (*objeto*) enciende (*evento*) un automóvil (*objeto*) con lo que el carro pasa de un estado de inmovilidad a uno de movimiento. *Otro ejemplo*, es cuando una persona hace un click con el botón izquierdo del mouse de una computadora para que esta lleve a cabo una acción.

Clases

Es una plantilla o molde que permite la creación de un objeto.

Las clases presentan el estado de los objetos a los que representan mediante variables denominadas **atributos**. Cuando se instancia un objeto, el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece el objeto.

Con la **POO** es necesario aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de objetos, propiedades y métodos; pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo, vamos a pensar en un automóvil para tratar de modelizarlo en un esquema de POO. Diríamos que el automóvil es el elemento principal que tiene una serie de características como ser: el color, modelo o marca. Además, tiene una serie de funcionalidades asociadas, como ser: ponerse en marcha, parar o aparcar, etc.

Elementos de la Programación Orientada Objetos

Objeto

Es la entidad básica en tiempo de ejecución en un sistema orientado a objetos; este puede representar una persona, cuenta bancaria, tabla de alumnos o cualquier elemento que el sistema tenga que manipular.

Características de los Objetos

Por ser entes que pueden ser manipulados en el mundo real, los objetos tienen características que los distinguen. En este libro se exponen tres grandes elementos:

1. **Atributos:** Son características que definen al objeto; también son conocidas como *propiedades* y proporcionan información acerca del ente. Por ejemplo, una pizarra tiene varios atributos: su tamaño, color, forma, dimensiones, etc. No deben confundirse los atributos con sus valores. Muchas personas piensan que un atributo de una computadora es que es negra., sin embargo, la propiedad es el color y el valor de esta propiedad, es negro.
2. **Métodos:** Son operaciones que un objeto puede llevar a cabo para transformar el entorno en que se encuentran o para comunicarse con otros objetos. Un automóvil posee un método de arranque, el cual le permite ponerse en marcha, así como posee un método de frenado que le permite detenerse. Un teléfono celular posee un método de emisión de señal, que le permite comunicarse con otros aparatos del mismo tipo.
3. **Eventos:** Es una acción llevada a cabo por un ente externo que modifica el estado del objeto. *Por ejemplo:* Una persona (*objeto*) enciende (*evento*) un automóvil (*objeto*) con lo que el carro pasa de un estado de inmovilidad a uno de movimiento. *Otro ejemplo,* es cuando una persona hace un click con el botón izquierdo del mouse de una computadora para que esta lleve a cabo una acción.

Clases

Es una plantilla o molde que permite la creación de un objeto.

Las clases presentan el estado de los objetos a los que representan mediante variables denominadas **atributos**. Cuando se instancia un objeto, el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece el objeto.

Las clases incluyen los datos y los códigos que operarán sobre esos datos. Una clase es la encargada de enlazar códigos y datos; C++ utiliza una especificación de una clase para la construcción de objetos.

Un buen ejemplo para comprender el concepto de clase, se muestra en el proceso de fabricación de ladrillos de piso. Para hacer un ladrillo de piso, la persona encargada prepara la mezcla y luego procede a tomar un molde con el que le dará forma al ladrillo. La persona encargada coloca el molde en un área plana, luego vierte sobre este la mezcla y espera a que esta endurezca; una vez que transcurrió el tiempo necesario, se extrae el molde y se obtiene el ladrillo. Si se observa detenidamente el ladrillo, se verá que este adquirió muchas de las propiedades del molde (tiene la misma forma y dimensión); sin embargo, el molde no puede ser utilizado como un ladrillo ya que sólo sirve para hacerlos.

En la POO el molde del ladrillo hace las veces de una clase. La clase es una plantilla que por sí misma no es un objeto, sin embargo, le confiere muchas de sus características a estos cuando son creados. El objeto puede tener sus propias características, no necesariamente iguales a las de su clase (el molde puede ser de metal o madera y el ladrillo de cemento, grava y arena) pero en el contexto global, son bastante parecidos.

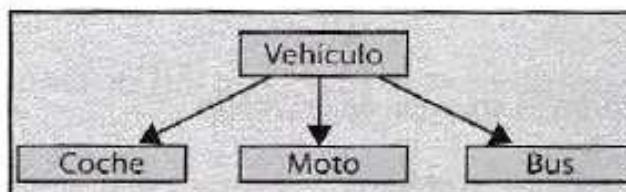
Abstracción de Datos y Encapsulamiento

A la envoltura de datos y funciones en una sola unidad llamada clase, se conoce como **encapsulamiento**. Los datos no son accesibles al mundo exterior y solamente aquellas funciones que están envueltas en la clase pueden acceder a ellos. A este aislamiento de los datos del acceso directo por el programa se le denomina **ocultación de los datos u ocultamiento de la información**.

La abstracción se refiere al acto de representar las características esenciales sin incluir los detalles de fondo o las explicaciones.

Herencia

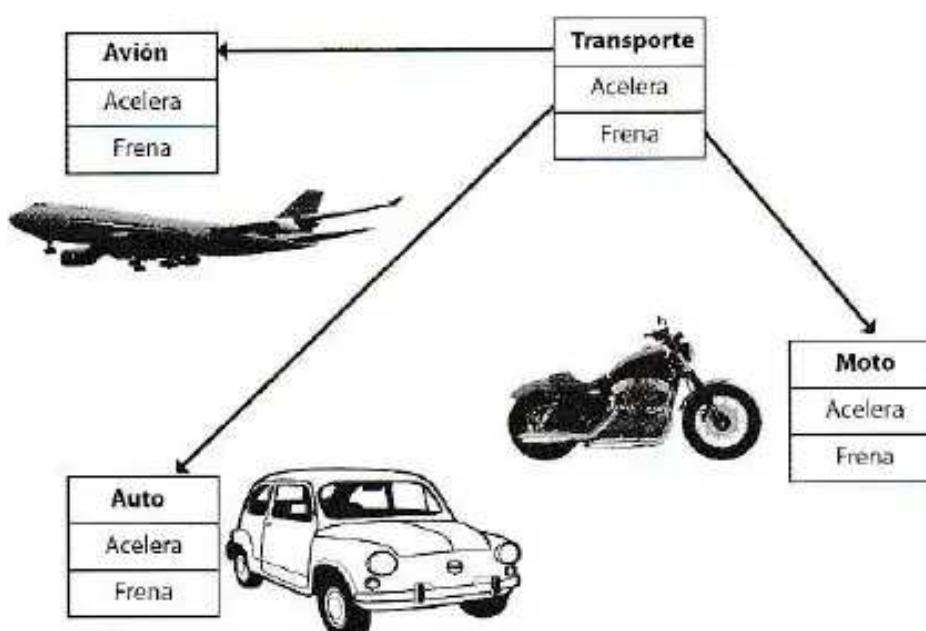
Es la propiedad que permite a los objetos crearse a partir de otros objetos. También podemos decir, que es el proceso por el que los objetos de una clase adquieren las propiedades de los objetos de otra clase. En POO, el concepto de herencia suministra la idea de reusabilidad. Esto significa, que podemos añadir características a una clase existente sin modificarla.



Polimorfismo

Es un vocablo griego que significa la posibilidad de tomar más de una forma. Se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

En algunos lenguajes, el término polimorfismo es también conocido como **Sobrecarga de parámetros** ya que las características de los objetos permiten aceptar distintos parámetros para un mismo método (diferentes implementaciones) generalmente, con comportamientos distintos e independientes para cada una de ellas.



Pase de Mensajes

Un programa orientado a objetos consta de un conjunto de objetos que se comunican entre sí enviando y recibiendo información de la misma forma que las personas se pasan mensajes entre ellas.

Un mensaje para un objeto es una solicitud para la ejecución de un procedimiento, y por lo tanto, invocará a una función (procedimiento) en el objeto receptor que genera el resultado deseado. El pase del mensaje implica la especificación del nombre del objeto, el nombre de la función (mensaje) y la información a enviar.

El proceso de programar en un lenguaje orientado a la POO involucra los siguientes pasos básicos:

1. Creación de las clases que definen los objetos y su comportamiento.
2. Creación de los objetos a partir de las definiciones de clases.
3. Establecimiento de la comunicación entre objetos.

Estrategia de Aprendizaje # 2

Instrucciones: En parejas, investigar y realizar cada una de las actividades que se le muestran a continuación:

1. ¿Qué es una función?
2. ¿Qué es un parámetro?
3. ¿Cuál es la utilidad de las funciones?
4. Crear una función que devuelva como resultado el impuesto y descuento.
5. Crear una función que devuelva como resultado la suma de dos números.
6. Ejemplos de parámetros por valor y por referencia.

Creación de Clases

Al definir una clase, se realiza la declaración de los datos que esta contiene y el código que opera en esos datos. Existen clases de orden simple que pueden contener solamente código o solamente datos, pero por lo general, la mayoría de las clases contienen ambos. En conjunto, los datos se almacenan en las variables y el código en las funciones. A las funciones y variables que forman una clase se les llama *miembros de la clase*.

Una clase tiene un conjunto de datos y operaciones que deben estar unidas de la siguiente manera:



Las clases se crean con la palabra *class*, el objetivo de crear una clase es la creación de nuevos tipos abstractos de datos, y con esto entra en juego el paradigma de programación orientada a objetos.

Sintaxis:

```
Class nombre_clase
{
    Private:
        Declaración de variables;
        Declaración de funciones;
    Public:
        Declaración de variables;
        Declaración de funciones;
};
```

Existen 3 tipos de usuarios de una clase:

- La propia clase
- Usuarios genéricos
- Clases derivadas

Cada uno de ellos tiene acceso a las clases con palabras reservadas tales como:

- **Private:** Para la propia clase.
- **Public:** Para usuarios genéricos.
- **Protected:** Para clases derivadas.

Ejemplo: Crear una clase que almacene los datos de una persona.

```
Persona.cpp - X
Person - Person - 
// Persona.cpp:
// 

#include "stdafx.h"
class Persona
{
    char nombre[50]; //Propiedad privada
public: int edad; //Propiedad pública
};

int main()
{
    return 0;
}
```

Como se puede observar, en el ejemplo anterior se crearon dos propiedades que son nombre y edad; nombre es privada y por lo tanto, no se puede acceder desde afuera de la clase, edad es pública y se puede acceder desde adentro o afuera de la clase. Esta clase también podría tener más propiedades como: color_pelo, color_piel, manos, etc.

Crear Objetos

Una vez declarada una clase se puede declarar variables de ese tipo utilizando el nombre de la clase seguido del nombre de la variable.

Ejemplo:

Persona x;

En este caso, se creó la variable "x" de tipo persona. En C++ las variables de clase se denominan **objetos**; por lo que podemos decir que "x" es un objeto de tipo persona. También se pueden declarar más variables de tipo persona.

Acceso a Miembros de una Clase

Es importante recordar, que los datos privados de una clase solamente pueden ser accesibles mediante las funciones miembros de la clase a la cual pertenecen y los datos públicos son accesibles tanto de afuera como de adentro de la clase a la cual pertenecen.

Ejemplo:

```

Editor Persona.cpp - X
Persono - (Ámbito global) + [O: main()]
// Persona.cpp
//

#include "stdafx.h"
using namespace std;
class Persona
{
    char nombre[50]; //Propiedad privada
public: int edad; //Propiedad pública
};

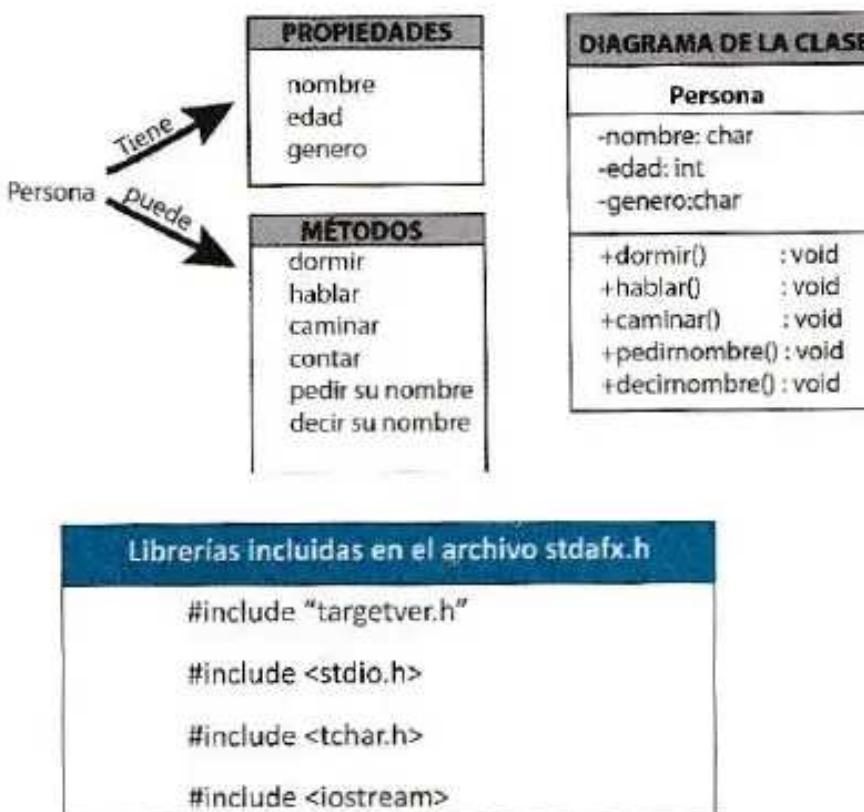
int main()
{
    Persona x; //Creación del objeto
    x.edad = 20;
    cout << "Su edad es: " << x.edad << endl;
    system("pause");
    return 0;
}

```

Como podemos observar, en el ejemplo anterior se creó una instancia de la clase **Persona** con la sentencia Persona x;, una vez instanciado se puede utilizar el objeto "x" colocando el nombre de la variable seguido de un punto y la propiedad o función con la cual se desea interactuar como se realizó en el ejemplo con la sentencia x.edad =20;.

Ejemplos:

1. Crear una clase que permita solicitar el nombre, edad y género; que a su vez, también sea capaz de decir el nombre, hablar, dormir, etc.

**Código del Programa**

```
// ClasePedirnombre.cpp:
#include "stdafx.h"
using namespace std;
class Persona
{
public:
    void dormir();
    void hablar();
    void caminar();
    void pedirnombre();
    void decirnombre();
private:
    char nombre[50];
    int edad;
```

```
char genero[2];
};

void Persona::dormir()
{
    cout << "zzzzzzzzzzzzzzzzzz" << endl;
}

void Persona::hablar()
{
    cout << "Hola, hola, hola, hola" << endl;
}

void Persona::caminar()
{
    cout << "Estoy caminando" << endl;
}

void Persona::pedirnombre()
{
    cout << "Ingresar mi nombre: "; cin.getline (nombre,50); cout << endl;
    cout << "Ingresar mi edad: "; cin >> edad; cout << endl;
    cout << "Ingresar mi g\x82nero: "; cin >> genero; cout << endl;
}

void Persona::decirnombre()
{
    cout << "Mi nombre es: " << nombre << endl;
    cout << "Tengo: " << edad << " a\xxa4os " << endl;
    //if (strcmp(palabra1, palabra2) == 0) {
        if (strcmp(genero,"F") == 0 || strcmp(genero, "f") == 0)
    {
        cout << "Soy Mujer" << endl;
    }
}
```

```
}

else

{

    cout << "Soy Hombre" << endl;

}

int main()

{

    Persona per1;

    per1.caminar();

    per1.pedirnombre();

    per1.decirnombre();

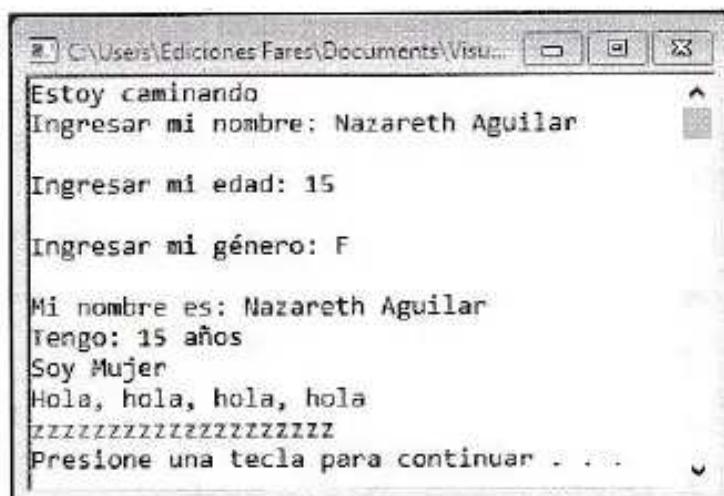
    per1.hablar();

    per1.dormir();

    system("pause");

    return 0;

}
```



2. Crear un programa que permita realizar la suma y resta de dos números, utilizando para tal propósito la programación orientada a objetos.

Librerías incluidas en el archivo stdafx.h

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
```

Código del Programa

```
// SumayRestaClase.cpp:
#include "stdafx.h"
using namespace std;
class operacionessumaresta
{
public:
    double operacionessumaresta::resta(double n1, double n2)
    {
        return n1 - n2;
    }
    double operacionessumaresta::suma(double n1, double n2)
    {
        return n1 + n2;
    }
};
int main()
{
    operacionessumaresta calculos;
    double numero1 = 0, numero2 = 0;
    double suma1, resta1;
    //Solicitar los dos números
    cout << "Introduzca el primer número: "; cin >> numero1; cout << endl;
    cout << "Introduzca el segundo número: "; cin >> numero2; cout << endl;
    //Realizar los cálculos
    suma1 = calculos.suma(numero1, numero2);
```

```
resta1 = calculos.resta(numero1, numero2);

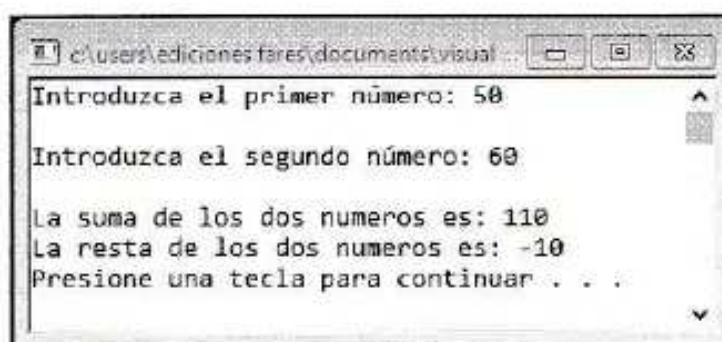
//Imprimir el resultado de los cálculos

cout << "La suma de los dos numeros es: "
     << suma1 << endl;

cout << "La resta de los dos numeros es: "
     << resta1 << endl;

system("pause");

return 0;
}
```



Características de las Clases en C++

- **Nombre de la Clase:** Sirven para identificar todos los objetos que tengan características comunes.
- **Conjunto de Atributos o Miembros:** Representan el estado de cada objeto.
- **Conjunto de Métodos o Funciones Miembro:** Permiten que los objetos cambien de estado dependiendo del estado anterior que tuviera el objeto.
- **Niveles de Acceso:** Para proteger ciertos miembros de la clase, normalmente se definirán como ocultos (**Private**) los atributos y visibles (**Public**) los métodos.

Atributos o Datos Miembros de una Clase

Como mencionamos anteriormente, son los descriptores del estado de un objeto, un atributo consta de dos partes: **nombre del atributo y valor**.

Los atributos de una clase pueden ser:

- De cualquier tipo básico (int, float, booleanos)
- De tipo estructura (Struct)
- Un arreglo
- Un objeto de otra clase

Puntos Claves de la Programación Orientada a Objetos que un Programador no debe ignorar

- Los atributos de una clase son privados.
 - Los métodos de una clase son públicos.
 - A lo privado de una clase sólo se puede acceder desde el código de los métodos.
 - Objeto es a clase como variable es a tipo.
 - Un atributo es aquel que describe el estado de un objeto.
 - Los métodos describen el comportamiento de los objetos de una clase.
 - El paso de mensajes hace que un objeto se manipule.
 - Interfaz son todas las operaciones y valores visibles desde el exterior de la clase.

Anotaciones importantes al final del estudio de la unidad:

Autoevaluación # 2

Tipo Verdadero o Falso

Instrucciones: Escribir en el paréntesis de la derecha una V en caso que considere la proposición como verdadera o una F en caso que la considere falsa, justifique su respuesta en caso de ser falsa.

1. Stdlib.h pertenece al conjunto de librerías estándar de C++..... ()

2. Al declarar una variable tipo Int en C++ nos estamos refiriendo a enteros..... ()

3. La codificación es la conversión de un programa en algoritmo..... ()

4. C++ nació como una extensión del lenguaje C..... ()

5. Los comentarios se utilizan para ayudar al programador a recordar para qué se utilizó una línea de código..... ()

6. La librería iostream.h, se utiliza para trabajar con fechas y horas..... ()

7. La función principal dentro de un programa main() indica dónde comienza y finaliza un programa..... ()

Programación I

Tipo Enumeración

Instrucciones: Enlistar en forma clara y ordenada lo que continuación se le pide:

1. Enliste las partes que debe contener todo programa creado en C++:

- a) _____
- b) _____
- c) _____
- d) _____
- e) _____
- f) _____

2. Enumere 5 tipos de variables definidas en C++:

- a) _____
- b) _____
- c) _____
- d) _____
- e) _____

3. Enliste los operadores aritméticos que intervienen en el lenguaje C++:

- a) _____
- b) _____
- c) _____
- d) _____
- e) _____
- f) _____

4. Cite 10 librerías estándar de C++

- a) _____
- b) _____
- c) _____
- d) _____
- e) _____
- f) _____

g) _____

h) _____

i) _____

j) _____

5. Enliste 5 secuencias de escape:

a) _____

b) _____

c) _____

d) _____

e) _____

Programación I

Tipo Respuesta Breve

Instrucciones: Conteste en forma clara y ordenada cada una de las interrogantes que se le muestran a continuación:

1. ¿Cuál es el objetivo de declarar variables?

2. ¿Qué es un identificador?

3. ¿Cuáles son las reglas que se deben seguir para formar los identificadores en C++?

4. ¿Qué es un operador en C++?

5. ¿Cuáles son los operadores lógicos utilizados en C++?

6. ¿Para qué se utilizan los operadores de asignación?

7. ¿Con qué carácter se identifica el final de una sentencia en C++?

8. ¿Cuál es la función de las secuencias de escape?

9. ¿Qué es Visual C++?

Tipo Práctico

Instrucciones: Resuelva los siguientes ejercicios:

- Crear un programa con menú en donde el usuario introduzca una medida de longitud especificando si está en centímetros, kilómetros o pulgadas y se realice la conversión a metros utilizando **Switch-Case**.
- Crear un programa con menú inventado por el lector utilizando **Switch-Case**.
- Se necesita un programa que permita manejar transacciones de una cuenta.
 - El saldo inicial de la cuenta debe ser de Lps. 0.00.
 - El programa debe solicitar al usuario que indique si desea realizar un depósito o un retiro.
 - Si el usuario elige hacer un retiro, se solicita un valor y debe verificarse que haya saldo suficiente para retirar. De no ser así, se envía un mensaje al usuario notificando esa situación. Si hay saldo suficiente, se resta el valor ingresado al saldo.
 - Si el usuario elige hacer un depósito se solicita un valor y ese valor se suma al saldo.
 - Al final de cada transacción se pregunta al usuario si desea realizar otra transacción. Si contesta afirmativamente, se repiten las acciones anteriores; si no, se termina el programa, mostrando el saldo final de la cuenta.
- Se necesita un programa que permita la captura de cinco valores correspondientes a radios de círculos.

Para cada uno de los cinco valores se requiere que se calcule y muestre en pantalla los siguientes datos del círculo:

- *Diámetro*: Se calcula multiplicando el radio por 2.
- *Circunferencia*: Se calcula multiplicando el diámetro por π (3.1416).