UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM TEXCOCO

INGENIERÍA EN COMPUTACIÓN

1ER SEMESTRE

EJERCICIOS DE PROGRAMACIÓN ESTRUCTURADA

M. en I.S.C. IRENE AGUILAR JUÁREZ

AGOSTO 2015

C O N T E N I D O

Compendio y Presentación:

Las tareas de un Ingeniero en computación son variadas pero es indudable que las tareas de programación son inherentes a su desempeño laboral, las organizaciones se enfrentan a problemas en su vida cotidiana que pueden ser resueltos con apoyo del Ingeniero y las aplicaciones informáticas adecuadas.

Las computadoras son herramientas que automatizan el procesamiento de la información, el manejo de los lenguajes de programación permite que los Ingenieros programen aplicaciones sencillas pero eficientes pensadas y generadas a medida de las organizaciones.

Este material aborda los primeras y más complejas competencias que el alumno debe adquirir en un curso de programación, consiste en 17 temas que se podrán realizar durante el curso, facilitando al alumno y al docente el tratamiento de los conceptos básicos de programación, estas temas están encaminadas a la adquisición de las habilidades en el alumno para programar soluciones computacionales, estas habilidades son las siguientes:

1. Identificar las fases de la metodología de programación estructurada para la solución de problemas
2. Aplicar la programación estructurada en la solución de problemas utilizando lenguaje informal y diagramas de flujo
3. Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo
4. Utilizar arreglos unidimensionales y bidimensionales para el almacenamiento de datos en la solución de problemas

A gran detalle los temas acompañan a alumno en el análisis de problemas, la propuesta de algoritmos y su especificación en notación algorítmica. Posteriormente se le dan a conocer las reglas de sintaxis del lenguaje de programación C y se le guía paso a paso en la codificación de programas usando sentencias secuenciales, condicionales y cíclicas. Los temas están acompañados de ejercicios y de las reglas de evaluación para ser aplicables en la evaluación continua del alumno.

Se anexan actividades complementarias útiles al profesor como alternativa en el trabajo de evaluación de los alumnos o para que sea usada como autoevaluación del alumno.

Tema 1: Aplicación de la metodología de Resolución de Problemas

U. Competencia 1

Duración Estimada: 1 sesión de 50 minutos

Competencia: Identificar las fases de la metodología de programación estructurada para la solución de problemas

Objetivo: El alumno aplicará la metodología para resolver problemas.

Introducción Teórica:

La resolución de problemas es una parte clave de los cursos de ingeniería, y también de los de ciencias de la computación, matemáticas, físicas y química. Por lo tanto, es importante tener una estrategia consistente para resolver los problemas. También es conveniente que la estrategia sea lo bastante general como para funcionar en todas estas áreas distintas.

La metodología para resolver problemas que usaremos tiene cinco pasos.

1. **Plantear el problema claramente.**
2. **Describir la información de entrada y salida.**
3. **Resolver el problema a mano (o con una calculadora) para un conjunto de datos sencillo.**
4. **Solución**
5. **Probar el programa con diversos datos**.

Analizaremos cada uno de estos pasos con un ejemplo sencillo. “Suponga que hemos recabado una serie de temperaturas de un sensor de cierto equipo que se está usando en un experimento. Se tomaron mediciones de temperatura cada 30 segundos, durante 5 minutos, en el curso del experimento. Queremos calcular la temperatura media y también graficar los valores de temperatura.”

1. **PLANTEAMIENTO DEL PROBLEMA**

El primer paso es plantear el problema claramente. Es en extremo importante preparar un enunciado claro y conciso del problema para evitar cualquier malentendido. Para el ejemplo, el enunciado del problema es el siguiente:

“Calcular la media de una serie de temperaturas. Graficar los valores de tiempo y temperatura”.

2. DESCRIPCIÓN DE ENTRADAS/SALIDAS

El segundo paso consiste en describir cuidadosamente la información que se da para resolver el problema y luego identificar los valores que se deben calcular. Estos elementos representan las

entradas y salidas del problema y pueden llamarse colectivamente entrada/salida o E/S. En muchos problemas resulta útil hacer un diagrama que muestre las entradas y salidas. En este punto, el programa es una "abstracción" porque no estamos definiendo los pasos para determinar las salidas; sólo estamos mostrando la información que se usará para calcular la salida.

Éste es el diagrama de E/S para el presente ejemplo: valores de tiempo promedio de temperaturas valores de temperatura gráfica de los valores de tiempo y temperatura.

3. **EJEMPLO A MANO**

El tercer paso es resolver el problema a mano o con una calculadora, empleando un conjunto sencillo de datos. Se trata de un paso muy importante y no debe pasarse por alto, ni siquiera en problemas sencillos. Éste es el paso en que se detalla la solución del problema. Si no podemos tomar un conjunto sencillo de números y calcular la salida, no estamos preparados para continuar con el siguiente paso; debemos releer el problema y tal vez consultar material de referencia.

Para este problema, el único cálculo consiste en calcular la media de una serie de valores de temperatura. Supongamos que usamos los siguientes datos para el ejemplo a mano:

Tiempo (minutos)	Temperatura (grados ºF)
0.0	105
0.5	126
1.0	119

Calculamos a mano la media como (105+126+119)/3

4. SOLUCIÓN

Una vez que podamos resolver el problema para un conjunto sencillo de datos, estamos listos para desarrollar un ***algoritmo***: un bosquejo paso a paso de la solución del problema. Si el problema es complejo puede ser necesario escribir a grandes rasgos los pasos y luego descomponer esos pasos en otros más pequeños. En este paso estamos preparados para realizar el programa correspondiente.

5. PRUEBA

El paso final de nuestro proceso de resolución de problemas es probar la solución. Primero debemos probar la solución con los datos del ejemplo a mano porque ya calculamos la solución antes.

Procedimiento

1. Aplica los 5 pasos antes mencionados para proponer una solución al siguiente problema:

   Es necesario conocer el número de kilómetros que avanzará un automóvil conociendo la velocidad a la que se mueve expresado en kilómetros por hora y el tiempo que dura su trayectoria

2. Documenta la aplicación de los cincos pasos explicados anteriormente llenando el siguiente cuadro:

1. Plantear el problema claramente.	
2. Describir la información de entrada y salida.	
3. Resolver el problema a mano (o con una calculadora) para un conjunto de datos sencillo.	
4. Solución	
5. Probar el programa con diversos datos.	

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Aplicación correcta de los pasos en el problema identificado	Entrega a tiempo	Calificación	Firma o sello

Evidencia: Cuadro terminado y practica calificada por el profesor

Tema 2: Diagramación de Algoritmos Computacionales

U. Competencia 2

Duración Estimada: 2 sesión de 50 minutos

Competencia : Aplicar la programación estructurada en la solución de problemas utilizando lenguaje informal y diagramas de flujo

Objetivo: El alumno aplicará las reglas para especificar algoritmos usando Diagramas de Flujo

Introducción Teórica:

Cuando se inicia en el mundo de la programación de computadoras, es necesario a la hora de construir algoritmos primero que todo intentar plantear al nivel de símbolos dicha solución. A la construcción con los símbolos indicados es a lo que se conoce con el nombre de Diagrama de flujo. **Recuerde que un algoritmo es la secuencia lógica de pasos que se sigue en la solución de un problema determinado.**

La palabra algoritmo proviene del nombre del matemático persa del siglo IX Muhammad ibn Musa al-Jwarizmi, un algoritmo es una secuencia ordenada de pasos, exenta de ambigüedad, que permite la resolución de un problema determinado

Características de un algoritmo
Preciso: exento de ambigüedad
Finito: si se sigue el algoritmo, se debe terminar en algún momento
Definido: si se sigue dos veces el algoritmo con los mismos datos de entrada, la salida debe ser la misma.

Representación de un algoritmo
Método informal
- Lenguaje natural
  - Ventajas: comprensible e intuitivo
  - Inconveniente: impreciso

Métodos formales
- Pseudocódigo
- Lenguaje natural limitado y sin ambigüedad
- Diagramas
- Diagramas de flujo u organigramas

Utiliza un conjunto de símbolos para representar cada estructura de control y mediante líneas de flujo se indica el orden en que se realiza el flujo lógico del algoritmo

Diagramas de Nassi-Schneiderman o Diagramas N-S
Los pasos sucesivos se escriben en cajas con distintas formas según la estructura de control que representen

Los diagramas de flujo (o flujo gramas) son diagramas que emplean símbolos gráficos para representar los pasos o etapas de un proceso. También permiten describir la secuencia de los distintos pasos o etapas y su interacción.
Las personas que no están directamente involucradas en los procesos de realización del producto o servicio, tienen imágenes idealizadas de los mismos, que pocas veces coinciden con la realidad.
La creación del diagrama de flujo es una actividad que agrega valor, pues el proceso que representa está ahora disponible para ser analizado, no sólo por quienes lo llevan a cabo, sino también por todas las partes interesadas que aportarán nuevas ideas para cambiarlo y mejorarlo.

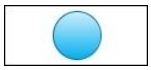
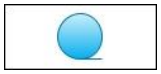
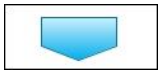
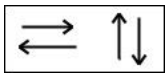
Ventajas de los Diagramas de Flujo
Favorecen la comprensión del proceso a través de mostrarlo como un dibujo. El cerebro humano reconoce fácilmente los dibujos. Un buen diagrama de flujo reemplaza varias páginas de texto.
Permiten identificar los problemas y las oportunidades de mejora del proceso. Se identifican los pasos redundantes, los flujos de datos y los puntos de decisión.
Muestran las interfaces cliente-proveedor y las transacciones que en ellas se realizan, facilitando a los empleados el análisis de las mismas.
Son una excelente herramienta para capacitar a los nuevos empleados y también a los que desarrollan la tarea, cuando se realizan mejoras en el proceso.

Desarrollo del Diagrama de Flujo
Las siguientes son acciones previas a la realización del diagrama de flujo:
1. Identificar los datos necesarios para el diagrama de flujo.
2. Definir que se espera obtener del diagrama de flujo.
3. Establecer el nivel de detalle requerido.
4. Determinar los límites del proceso a describir.

Los pasos a seguir para construir el diagrama de flujo son:
1. Establecer el alcance del proceso a describir. De esta manera quedará fijado el comienzo y el final del diagrama. Frecuentemente el comienzo es la salida del proceso previo y el final la entrada al proceso siguiente.
2. Identificar y listar las principales actividades/subprocesos que están incluidos en el proceso a describir y su orden cronológico.
3. Si el nivel de destalle definido incluye actividades menores, listarlas también.
4. Identificar y listar los puntos de decisión.
5. Construir el diagrama respetando la secuencia cronológica y asignando los correspondientes símbolos.
6. Asignar un título al diagrama y verificar que esté completo y describa con exactitud el proceso elegido.

Símbolos y Reglas para dibujar un diagrama de flujo.
Los símbolos tienen significados específicos y se conectan por medio de flechas que indican el flujo entre los distintos pasos o etapas.
Los Diagramas de flujo se dibujan generalmente usando algunos símbolos estándares; sin embargo, algunos símbolos especiales pueden también ser desarrollados cuando sean

requeridos. Algunos símbolos estándares, que se requieren con frecuencia para diagramar programas de computadora se muestran a continuación:

	Inicio o fin del programa
	Pasos, procesos o líneas de instrucción de programa de computo
	Operaciones de entrada
	Toma de decisiones y Ramificación
	Conector para unir el flujo a otra parte del diagrama
	Cinta magnética
	Disco magnético
	Conector de pagina
	Líneas de flujo
	Anotación
	Display, para mostrar datos
	Envía datos a la impresora

Dentro de los símbolos fundamentales para la creación de diagramas de flujo, los símbolos gráficos son utilizados específicamente para operaciones aritméticas y relaciones condicionales. La siguiente es una lista de los símbolos más comúnmente utilizados:

+	Sumar	**>=**	Mayor o igual que
-	Menos	**<=**	Menor o igual que
*****	Multiplicación	**<>**	Diferente de
/	División		Si
±	Mas o menos		No
=	Equivalente a		True
>	Mayor que		False
<	Menor que		

Reglas para la creación de Diagramas

Los Diagramas de flujo deben escribirse de arriba hacia abajo, y/o de izquierda a derecha.

Los símbolos se unen con líneas, las cuales tienen en la punta una flecha que indica la dirección que fluye la información procesos, se deben de utilizar solamente líneas de flujo horizontal o verticales (nunca diagonales).
Se debe evitar el cruce de líneas, para lo cual se quisiera separar el flujo del diagrama a un sitio distinto, se pudiera realizar utilizando los conectores. Se debe tener en cuenta que solo se van a utilizar conectores cuando sea estrictamente necesario.
No deben quedar líneas de flujo sin conectar
Todo texto escrito dentro de un símbolo debe ser legible, preciso, evitando el uso de muchas palabras.
Todos los símbolos pueden tener más de una línea de entrada, a excepción del símbolo final.
Solo los símbolos de decisión pueden y deben tener más de una línea de flujo de salida.

En los siguientes ejercicios se verá la forma de aplicar la simbología de los diagramas de flujo en la solución de problemas de tipo matemático sencillos. Aunque parezcan sencillos es una manera de adentrar un poco en el conocimiento de la diagramación estructurada. En dichos ejercicios se utilizarán funciones que permitan la entrada de datos por el teclado como medio estándar de entrada de datos de la máquina y funciones que me permitan manejar la pantalla como medio de salida más utilizado en las computadoras para presentar información.

Adicional a las funciones de entrada y salidas de datos se utilizará la estructura de asignación en la solución de pequeñas expresiones de tipo matemático.

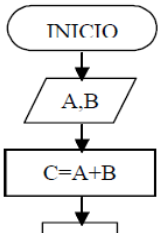
1. Desarrolle un algoritmo que le permita leer dos valores y escribir la suma de los dos.

Análisis:
Para dar solución a este ejercicio es necesario realizar tres tareas: Leer los valores que para el caso concreto del ejemplo son dos (2), Calcular la suma de dichos valores y por último escribir el resultado obtenido de dicha suma. Cada una de las tareas planteadas se en marcan dentro de un símbolo utilizado en diagramación así:

La lectura de cada dato desde el teclado se almacenará en variables, una guardará el primer valor que para el desarrollo se ha identificado con el nombre A y la otra el guardará el segundo valor que se ha denominado o identificado con el nombre B (estas operaciones se representarán en el símbolo de lectura. También se puede representar cada operación en símbolos aparte).

El cálculo de la suma se realizará y su valor será almacenado en la variable identificada como C (esta operación se representará en el símbolo de proceso) y por último el valor de respuesta almacenado en la variable C, se escribirá en la pantalla (esta operación se representa en el símbolo de escritura).
Vea el diagrama siguiente.

Diagrama de Flujo



Procedimiento

Analice los problemas planteados en el cuadro izquierdo y proponga los algoritmos que solucionen dichos problemas en el cuadro de la derecha aplicando las reglas para elaborar Diagramas de Flujo

1. Se necesita un programa que pueda leer los 3 lados de un triangulo y calcule el perímetro de la figura.	
2. Se necesita calcular el precio a pagar por 5 productos de precio diferente y aplicar el 10% de impuesto.	
3. Es necesario un programa que calcule el pago para un empleado con base a su sueldo base + un bono por desempeño que consiste en el 15% de su sueldo base.	

4. Elabore un programa que a partir del dato que represente un radio se calcule el área de un circulo	
5. Es necesario un programa que calcule el número de días vividos de una persona a partir de los datos fecha de nacimiento y fecha actual	

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Aplicación correcta de las reglas para un DF	Entrega a tiempo	Calificación	Firma o sello

Evidencia: Diagramas de Flujo terminado y practica calificada por al profesor

Tema 3: Usar las reglas de uso de un Pseudocódigo

U. Competencia 2

Duración Estimada: 2 sesiones de 50 minutos

Competencia : Aplicar la programación estructurada en la solución de problemas utilizando lenguaje informal y diagramas de flujo

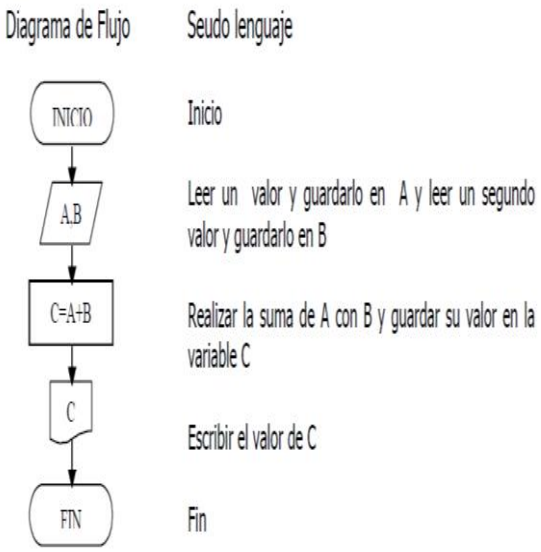
Objetivo: El alumno aplicará las reglas para especificar algoritmos usando Pseudocódigo

Introducción Teórica:

Pseudocódigo

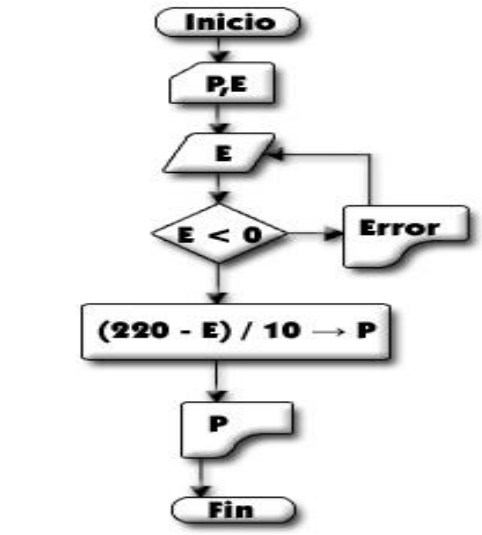
Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el Pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

El Pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El Pseudocódigo utiliza palabras que indican el proceso a realizar. .

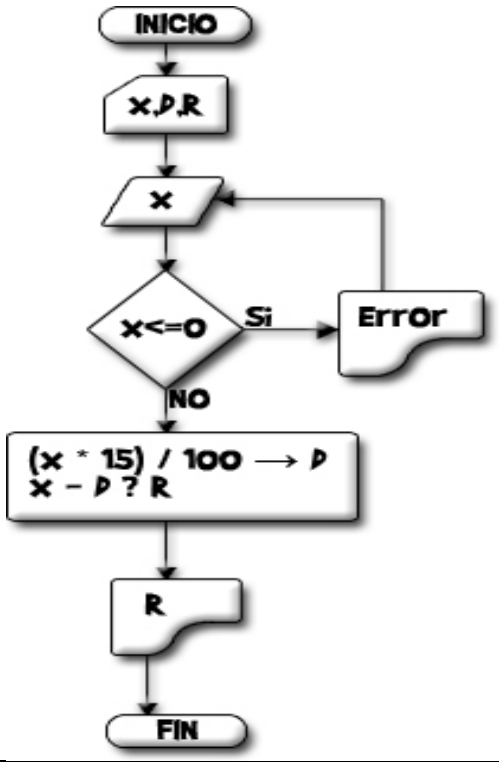


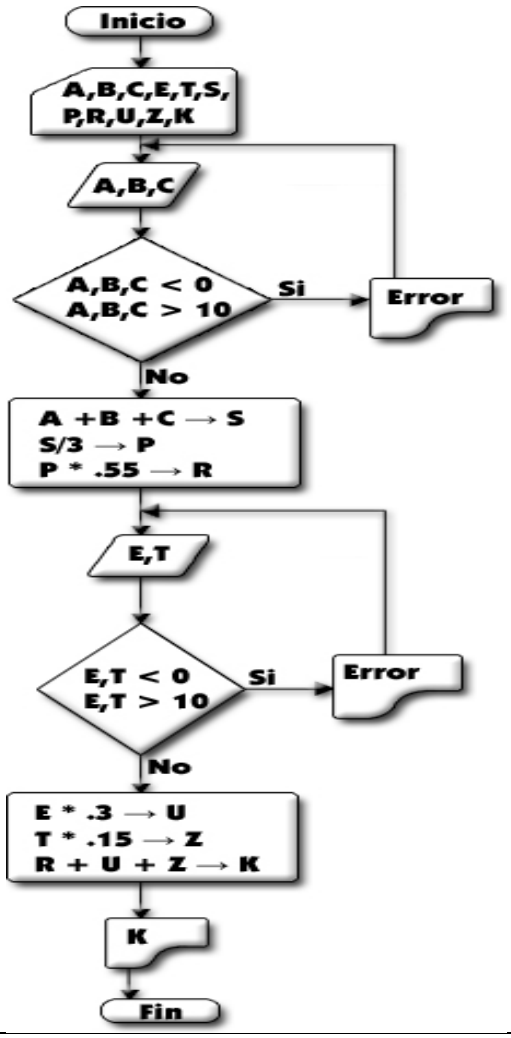
Procedimiento

Analice los Diagramas de Flujo propuestos y traslade el algoritmo a pseudocódigo


Inicio
P,E
E
E < 0
Error
(220 - E) / 10 → P
P
Fin

.


INICIO
X,P,R
X
X<=0
Si
Error
NO
(X * 15) / 100 → P
X - P ? R
R
FIN



Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Interpretación correcta de las reglas para un DF	Entrega a tiempo	Calificación	Firma o sello

Evidencia: Pseudocódigos terminados y practica calificada por el profesor

Tema 4: Conceptos básicos de la Programación

U. Competencia 3

Duración Estimada: 1 sesión de 50 minutos

Competencia : Codificar algoritmos en un lenguaje de programación estructurado

Objetivo: El alumno conocerá los conceptos básicos de la programación como introducción al léxico que usará en las siguientes prácticas de codificación

Introducción Teórica:

Para codificar los algoritmos será necesario usar un lenguaje de programación, de los cuales existen numerosos tipos en el ámbito computacional.

Los lenguajes de alto nivel suelen utilizar términos ingleses del tipo LIST, PRINT u OPEN como comandos que representan una secuencia de decenas o de centenas de instrucciones en lenguaje máquina. Los comandos se introducen desde el teclado, desde un programa residente en la memoria o desde un dispositivo de almacenamiento, y son interceptados por un programa que los traduce a instrucciones en lenguaje máquina.

Los programas traductores son de dos tipos: intérpretes y compiladores. Con un intérprete, los programas que repiten un ciclo para volver a ejecutar parte de sus instrucciones, reinterpretan la misma instrucción cada vez que aparece. Por consiguiente, los programas interpretados se ejecutan con mucha mayor lentitud que los programas en lenguaje máquina. Por el contrario, los compiladores traducen un programa íntegro a lenguaje máquina antes de su ejecución, por lo cual se ejecutan con tanta rapidez como si hubiesen sido escritos directamente en lenguaje máquina.

Se considera que fue la estadounidense Grace Hopper quien implementó el primer lenguaje de ordenador orientado al uso comercial. Después de programar un ordenador experimental en la Universidad de Harvard, trabajó en los modelos UNIVAC I y UNIVAC II, desarrollando un lenguaje de alto nivel para uso comercial llamado FLOW-MATIC. Para facilitar el uso del ordenador en las aplicaciones científicas, IBM desarrolló un lenguaje que simplificaría el trabajo que implicaba el tratamiento de fórmulas matemáticas complejas. Iniciado en 1954 y terminado en 1957, el FORTRAN (acrónimo de Formula Translator) fue el primer lenguaje exhaustivo de alto nivel de uso generalizado.

En 1957 una asociación estadounidense, la *Association for Computing Machinery* comenzó a desarrollar un lenguaje universal que corrigiera algunos de los defectos del FORTRAN. Un año más tarde fue lanzado el ALGOL (acrónimo de Algorithmic Language), otro lenguaje de orientación científica. De gran difusión en Europa durante las décadas de 1960 y 1970, desde entonces ha sido sustituido por nuevos lenguajes, mientras que el FORTRAN continúa siendo utilizado debido a las gigantescas inversiones que se hicieron en los programas existentes. El COBOL (acrónimo de Common Business Oriented Language) es un lenguaje de programación para uso comercial y empresarial especializado en la organización de datos y manipulación de archivos, y hoy día está muy difundido en el mundo empresarial.

El lenguaje BASIC (acrónimo de Código de Instrucciones Simbólicas de Uso General para Principiantes) fue desarrollado en el Dartmouth College a principios de la década de 1960 y está dirigido a los usuarios de ordenador no profesionales. Este lenguaje se universalizó gracias a la popularización de los microordenadores en las décadas de 1970 y 1980. Calificado de lento, ineficaz y poco estético por sus detractores, BASIC es sencillo de aprender y fácil de utilizar. Como muchos de los primeros microordenadores se vendieron con BASIC incorporado en el *hardware* (en la memoria ROM), se generalizó el uso de este lenguaje.
Aunque existen centenares de lenguajes informáticos y de variantes, hay algunos dignos de mención, como el PASCAL, diseñado en un principio como herramienta de enseñanza, hoy es uno de los lenguajes de microordenador más populares; el Logo fue desarrollado para que los niños pudieran acceder al mundo de la informática; el C, un lenguaje de Bell Laboratories diseñado en la década de 1970, se utiliza ampliamente en el desarrollo de programas de sistemas, al igual que su sucesor, el C++. El LISP y el PROLOG han alcanzado amplia difusión en el campo de la inteligencia artificial.

Procedimiento

a. Investiga los significados de los siguientes términos y forma un glosario que entregarás a tu profesor: Lista de Términos:
    1. Programa
    2. Programa fuente
    3. Programa objeto
    4. Programa ejecutable
    5. IDE
    6. Variable
    7. Constante
    8. Tipo de dato
    9. Palabra reservada
    10. Paradigma de programación
    11. Estructura de control
    12. Estructura cíclica
    13. Directiva de compilador
    14. Compilador
    15. Depurador
    16. Editor
    17. Interprete
    18. Función
    19. Parámetro
    20. Valor de retorno

Evaluación:

Se evaluará el desempeño del alumno con el glosario terminado y los criterios de evaluación serán los siguientes

Glosario ordenado alfabéticamente	Entrega a tiempo	Calificación	Firma o sello
Glosario en su portafolio de evidencias			

Tema 5: Estructura básica de un programa en lenguaje C

U. Competencia 3

Duración Estimada: 1 sesión de 50 minutos

Competencia : Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará la estructura básica para codificar un programa en lenguaje C

Introducción Teórica:

El siguiente ejemplo ilustra algunos de los componentes que con mayor frecuencia se usan cuando se trata de codificar programas utilizando el lenguaje de programación C. Cada una de las líneas es numerada para efectos de la explicación posterior que te dará tu profesor. En el momento de probarlo en el lenguaje de programación no es necesaria la numeración.

```
1. /* Programa para calcular el producto de dos números previamente
ingresados por teclado */
2. #include "stdio.h"
3. int a,b,c;
4. int producto(int x, int y);
5. main()
6. {
7. /* pide el primer número */
8. printf("digite un numero entre 1 y 100");
9. scanf("%d",&a);
10.
11. /* pide el segundo numero */
12. printf("digite un numero entre 1 y 100");
13. scanf("%d",&b);
14.
15. /* calcula y despliega el producto */
16. c = producto(a,b);
17. printf("\n%d por %d = %d", a, b, c);
18. }
19.
20. /* función que calcula y regresa el producto de sus dos argumentos */
21. int producto(int x, int y)
22. {
23. return(x*y);
24. }
```

Aunque cada uno de los programas es distinto, todos tienen características comunes. Los elementos básicos de un programa codificado en lenguaje C son los siguientes tomando el ejemplo anteriormente descrito:

- La función main() (línea 5 y desarrollada de la 6 – 18)
- La directiva #include (línea 2)
- Definición de variables (línea 3)
- Prototipo de función (línea 4)

- Enunciados del programa (línea 8, 9, 12, 13, 16, 17, 23)
- Definición de función (línea 21 – 24)
- Comentarios (Línea 1, 7, 11, 15, 20)

Procedimiento

2. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el ejemplo anterior, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar.

3. Con ayuda del entorno de desarrollo guarda el código con el nombre de Tema1, selecciona la ruta en la que desees guardarlo.

4. Usa el administrador de archivos para verificar que en la ruta solicitada se generó el programa fuente y el programa objeto, observa que los archivos son diferentes por su extensión, mientras el archivo fuente se almacena con la extensión .c o .cpp el programa objeto se genera con extensión .exe

5. Repite el proceso con el ejemplo 2

   a. Ejemplo2 :

```
#include<stdio.h>
#include<conio.h>
main()
{ char nombre[50]; int edad;
clrscr();
gotoxy(10,5); printf("¿Cómo te llamas?\n ");
scanf("%s",&nombre);
gotoxy(10,6); printf("¿Cuántos años tienes?\n");
scanf("%i", &edad);
edad = edad * 365;
gotoxy(10,8); printf("%s, has vivido %d días", nombre, edad);
gotoxy(40,22); printf("Pulsa cualquier tecla para terminar...");
getch(); }
```

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Ejemplo 1			
Ejemplo 2			

Evidencia: Archivo en el dispositivo de almacenamiento y práctica calificada por el profesor

Tema 6: Aplicación de reglas de sintaxis para identificadores

Duración Estimada: 1 sesión de 50 minutos

Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las reglas de sintaxis para la declaración de identificadores en el lenguaje C

Introducción Teórica:

Para codificar con el lenguaje C es necesario conocer y usar las reglas de sintaxis adecuadamente de lo contrario el depurador nos señalará varios errores de sintaxis con los cuales nos resultaría imposible lograr la compilación de nuestro programa fuente. A continuación se resumen las principales reglas y características de sintaxis del lenguaje C

Reglas para identificadores (Nombre de las variables)

1	El nombre no puede ser un apalabra reservada
2	Todo nombre debe iniciar con una letra o con el guión bajo(_)
3	El resto del nombre puede consistir de letras, guión bajo y/ o dígitos.
4	Solo se permiten letras , dígitos y guión bajo no se permiten otros caracteres, ni los acentos
5	Los 32 caracteres del identificador son significativos
6	Hay sensibilidad al tamaño; no es igual "a" que "A"

Tipo de dato simples o primitivos	Palabra o símbolo	cantidad de memoria
carácter	**char**	8 bits =1bytes
entero de longitud estándar	**int**	16 bits =2 bytes
entero de longitud grande	**long int**	32 bits = 4 bytes
entero de longitud corta	**short int**	16 bits =2 bytes
entero sin signo	**unsigned**	
punto flotante(real)	**float**	32 bits = 4 bytes
punto flotante doble precisión	**double**	64 bits= 8 bytes
punto flotante doble precisión grande	**long double**	80 bits =10 bytes
apuntadores	*****	

Ejemplo de identificadores:

```
int n, m, *p, v[100];
float r, a[10][10], *pf[100];
char c, *s, lin[80];
float superficie();
        struct naipe {
        int valor;
        char palo;
```

```
        } carta ;
enum colores{ rojo, azul,amarillo } color;
typedef struct {
        float real;
        float imaginaria;
        } complejo;   complejo c1,
```

Ejemplo para declarar variables:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int CUENTA;
char CADENA[20], A=48, B=49, C=50, NOMBRE[25];
float REAL=5.7;

main()
{
  printf("%d\n%s\n%c\n%c\n%c\n%s\n%f\n",CUENTA,CADENA,A,B,C,NOMBRE,REAL);}
```

Procedimiento

1. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el ejemplo anterior, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la practica en tu dispositivo de almacenamiento

2. De forma similar al ejemplo anterior y con la guía de tu profesor codifica un programa que te permita declarar las siguientes variables y que después los muestre en la pantalla

    a. Nombre
    b. Edad
    c. Dirección
    d. Sexo(M/F)
    e. Dinero necesario para pasajes al día
    f. Dinero necesario para comida al día

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Ejercicio 1			
Ejercicio 2			

Tema 7: Codificación para la entrada y salida de datos

Duración Estimada: 1 sesión de 50 minutos

Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las reglas de sintaxis básicas del lenguaje C para lograr la entrada y salida de datos.

Introducción Teórica:

La función printf() es la función que nos permite la muestra en pantalla de información asi como la función scanf() nos permite la entrada de los mismos. Su uso te fue explicado por el profesor y a continuación se te entrega la tabla que te informa en qué casos se debe usar el código de formato o la conversión de tipos de datos

Código de formato usando en la función printf y scanf

%d	Entero decimal
%lf	Flotante largo
%f	Número de punto flotante sin exponente
%e	Número de punto flotante con exponente
%x	Entero hexadecimal
%ld	Entero largo
%o	entero octal
%x	entero hexadecimal
%u	entero decimal sin signo
%s	cadena
%c	carácter sencillo

Conversión de Tipo
char<int<long<float<double
(tipo) variable ejemplo (int) 3.1415;

Con las reglas analizadas en la sección anterior codifica los siguientes algoritmos

Procedimiento

Codifica los siguientes ejemplos y almacénalos en tu dispositivo de almacenamiento para ejecutarlo y entregarlo al profesor

```
Ejemplo1
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
main()
{
 printf("Hola %c %d %s \n", '\t',15);
 printf("%-5.2f\n", 123.234);
 printf("%5.2f\n", 3.234);
 printf("%10s\n", "Hola");
 printf("%-.3s\n", "Hola");
 printf("%5.7f\n", 1.23456789);  }
```

Ejemplo 2

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int CUENTA;
char CADENA[20], A=48, B=49, C=50, NOMBRE[25];
float REAL;

main()
{
 scanf("%d", &CUENTA);        //Mete un entero decimal a la variable CUENTA
 scanf("%s", CADENA);   //Mete una cadena de caracteres al arreglo que
                        //empieza en la dirección CADENA
 scanf("%20s\n", CADENA);//Mete una cadena de 20 caracteres al arreglo que
                        //empieza en la dirección CADENA
 scanf("%c %c %c\n",&A,&B,&C);//Mete tres caracteres a las variables A, B y C
 scanf("%s", NOMBRE);         //Mete una cadena de caracteres al arreglo que
                        //empieza en la dirección NOMBRE
 scanf("%f", &REAL);//Mete un numero de punto flotante a la variable REAL
 printf("%d\n%s\n%c\n%c\n%c\n%s\n%f\n",CUENTA,CADENA,A,B,C,NOMBRE,REAL);
}
```

De forma similar a los ejemplos anteriores y con la guía de tu profesor codifica un programa que te permita declarar las siguientes variables y que después te permita almacenar valores para mostrarlos en la pantalla

a. Nombre
b. Edad
c. Dirección
d. Sexo(M/F)
e. Dinero necesario para pasajes al día
f. Dinero necesario para comida al día

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificación del ejemplo			
Codificación del ejercicio			

Tema 8: Solución de un problema un programa en lenguaje C

Duración Estimada: 1 sesión de 50 minutos

1. Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará la metodología de solución de problemas con apoyo de programas de estructura básica para codificar un programa en lenguaje C

Introducción Teórica:

Cuando se inicia en el fascinante mundo de la programación de computadoras, es necesario a la hora de construir algoritmos primero que todo intentar plantear al nivel de símbolos dicha solución. A la construcción con los símbolos indicados es a lo que se conoce con el nombre de Diagrama de flujo. **Recuerde que un algoritmo es la secuencia lógica de pasos que se sigue en la solución de un problema determinado.**

En los siguientes ejercicios se verá la forma de aplicar la simbología de los diagramas de flujo en la solución de problemas de tipo matemático sencillos.

Aunque parezcan sencillos es una manera de adentrar un poco en el conocimiento de la diagramación estructurada. En dichos ejercicios se utilizarán funciones que permitan la entrada de datos por el teclado como medio estándar de entrada de datos de la máquina y funciones que me permitan manejar la pantalla como medio de salida más utilizado en las computadoras para presentar información.

Adicional a las funciones de entrada y salidas de datos se utilizará la estructura de asignación en la solución de pequeñas expresiones de tipo matemático.

En los siguientes ejercicios se realiza el diseño de algoritmos al nivel de diagrama de flujo, pseudolenguaje, pseudo código y Código en el lenguaje Pascal y C, que dan solución a cada una de las preguntas planteadas.

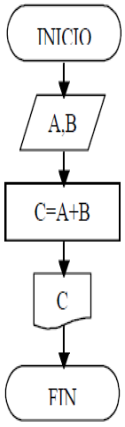
1. Desarrolle un algoritmo que le permita leer dos valores y escribir la suma de los dos.

Análisis:

Para dar solución a este ejercicio es necesario realizar tres tareas: Leer los valores que para el caso concreto del ejemplo son dos (2), Calcular la suma de dichos valores y por último escribir el resultado obtenido de dicha suma. Cada una de las tareas planteadas se en marcan dentro de un símbolo utilizado en diagramación así:

La lectura de cada dato desde el teclado se almacenará en variables, una guardará el primer valor que para el desarrollo se ha identificado con el nombre A y la otra el guardará el segundo valor que se ha denominado o identificado con el nombre B (estas operaciones se representarán en el símbolo de lectura. También se puede representar cada operación en símbolos aparte). El cálculo de la suma se realizará y su valor será almacenado en la variable identificada como C (esta operación se representará en el símbolo de proceso) y por último el valor de respuesta almacenado en la variable C, se escribirá en la pantalla (esta operación se representa en el símbolo de escritura). Vea el diagrama siguiente.

Diagrama de Flujo	Seudo lenguaje	

	Inicio
	
	Leer un valor y guardarlo en A y leer un segundo valor y guardarlo en B
	
	Realizar la suma de A con B y guardar su valor en la variable C
	
	Escribir el valor de C
	
	Fin
		#include "conio.h"
		#include "stdio.h"
		int A,B,C;
		main()
		clrscr();
		gotoxy(10,10);
		printf("Digite un número entero ");
		scanf("%d",&A);
		gotoxy(10,11);
		printf("Digite un número entero ");
		scanf("%d",&B);
		C= A+B;
		gotoxy(10,12);
		printf("La suma es : %d ",C);
		getch();
		}

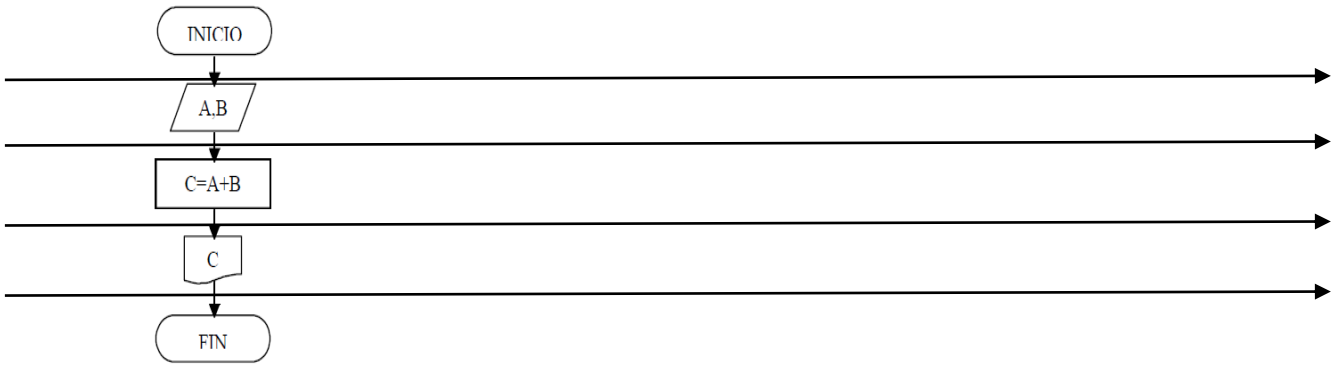
Procedimiento

1. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el ejemplo anterior, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la practica en tu dispositivo de almacenamiento

2. Analiza las líneas de código necesarias para desarrollar el algoritmo

Código en C

Diagrama de Flujo



Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificar Ejercicio 1			
Ejercicio 2			

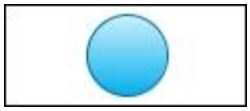
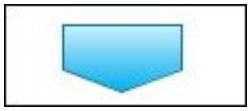
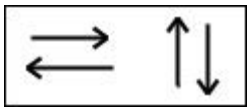
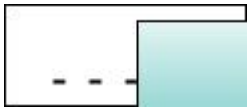
Tema 9: Codificación de un algoritmo con el lenguaje C

Duración Estimada: 1 sesión de 50 minutos

Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las reglas de sintaxis básicas del lenguaje C para codificar algoritmos.

Introducción Teórica:

Símbolo	Función del símbolo	Sintaxis en C
	Inicio o fin del programa	Para iniciar un programa en C usaremos la función **main(){ }**
	Pasos, procesos o líneas de instrucción de programa de computo	Para codificar los proceso del algoritmo se puede usar sentencias de asignación por ejemplo: **area=12*12; o sexo='M'; o nombre="Juan";**
	Operaciones de entrada	Para codificar la entrada de datos usaremos la función **scanf();**
	Toma de decisiones y Ramificación	Para cambiar el flujo de ejecución de acuerdo a una decisión simple se usará la estructura condicional **if ( )**
	Conector para unir el flujo a otra parte del diagrama	Este símbolo no es necesario codificarlo solo es para mejor comprensión del algoritmo
	Disco magnético	Este símbolo nos indica la lectura desde un archivo existente en la PC
	Conector de pagina	Este símbolo no es necesario codificarlo solo es para mejor comprensión del algoritmo
	Líneas de flujo	Determinan la secuenciación de las instrucciones
	Anotación	Este símbolo puedes codificarlo con los comentarios **//** de una línea **/* */** multi línea
	Envía datos a la impresora	Para codificar la salida de datos usaremos la función **printf(); ,** por el momento no se usará el acceso a otros dispositivos de salida

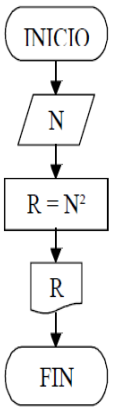
Display, para mostrar datos

Para codificar la salida de datos usaremos la función **printf();**

Procedimiento

1. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el siguiente ejemplo, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la practica en tu dispositivo de almacenamiento

Diagrama de flujo:



Seudo lenguaje:

Inicio

Leer el valor número entero y almacenarlo en la variable N.

Calcula el cuadrado del numero leído y que está almacenado en la variable N, y su resultado almacenarlo en la variable R.

Escriba el valor encontrado del cuadrado que esta almacenado en la variable R.

Fin

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificación del algoritmo			

Tema 10: Codificación de un algoritmo con condicional simple

Duración Estimada: 1 sesión de 50 minutos

1. Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las reglas de sintaxis básicas del lenguaje C para codificar algoritmos en los que se aplique la estructura **if () else.**

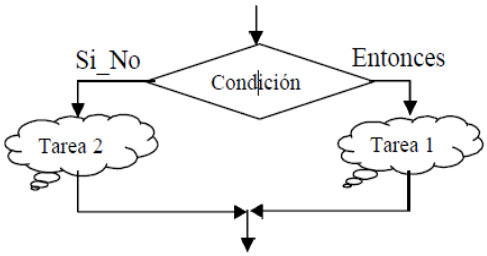
Introducción Teórica:

Cuando el programador desea especificar dos caminos alternativos en un algoritmo se deben utilizar estructuras de decisión. Una estructura de decisión dirige el flujo de un programa en una cierta dirección, de entre dos posibles, en función de un valor booleano. En lenguajes de programación estructurados la estructura condicional es la IF / ELSE. La cláusula ELSE en esta estructura es optativa. La forma en que trabaja esta sentencia resulta casi evidente a partir de la lógica de la lengua inglesa: Si (IF) la expresión booleana resulta cierta (TRUE), entonces la sentencia se ejecuta. Si la expresión booleana resulta falsa (FALSE), el control pasa a la siguiente (en orden descendente) instrucción del programa

Estructura Sencilla:

Forma general de uso:

```
Si (Condición) entonces
        Ejecuta bloque de instrucciones uno
Si_no
        Ejecuta bloque de instrucciones dos
Fin si
```



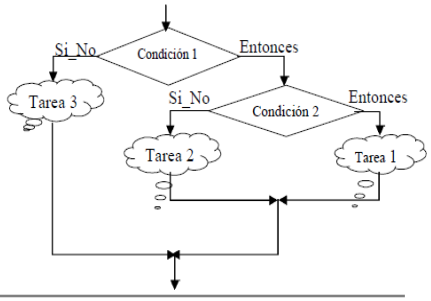
Una estructura de decisión puede estar anidada dentro de otra estructura de decisión. Hay que tener en cuenta que el anidamiento sea total. El inicio y el final de la estructura anidada debe quedar totalmente dentro del inicio y el final de la estructura que permite dicho anidamiento.

Se debe mantener el concepto que una estructura de decisión debe tener una sola entrada y una sola salida tanto para la estructura que anida como para la estructura anidada.

En el diagrama anterior se puede observar una estructura anidada del lado verdadero de la condición de la primera decisión.

Seudo lenguaje de la estructura de decisión anidada.

```
Si (Condición 1) entonces
        Si (Condición 2) entonces
                Ejecuta bloque de instrucciones tarea 1
        Si_no
                Ejecuta bloque de instrucciones tarea 2
        Fin_si
```



Si_no
    Ejecuta bloque de instrucciones tarea 3
Fin si.

Procedimiento

1. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el siguiente ejemplo, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la practica en tu dispositivo de almacenamiento

   Ejemplo: Calcular las posibles raíces para una ecuación de segundo grado: $ax^2+bx+c=0$

```
+-Algoritmo raíces
|   Variables reales a, b, c, x, y
|   Escribir  "Introduzca los coeficientes de mayor a menor grado."
|   Leer  a, b, c
| +-Si sqrt (b)>= 4*a*c entonces
| |   x=(-b + sqrt (b^2-4*a*c))/2a
| +-Sino
| |   Escribir "No existen raíces reales."
| +-Fin si
+-Final
```

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificación del algoritmo			

Tema 11: Uso de la librería math.h

Duración Estimada: 1 sesión de 50 minutos

1. Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las funciones incorporadas en la librería math.h con la finalidad de desarrollar programas matemáticos.

Introducción teórica

Para lograr programas complejos en el lenguaje C es necesario usar la inclusión de librerías que contengan funciones de uso específico y que puedan usarse para resolver problemas o cálculos. Una de las librerías usadas con mayor frecuencia en la librería math.h, a continuación se muestra la tabla con los datos de las funciones incluidas en la misma.

Función	Síntaxis	Descripción:
acos()	double acos(double x);	Calcula el valor principal del arco coseno de x. Puede producirse un error de dominio para los argumentos que no estén en el intervalo [-1, +1].
asin()	double asin(double x);	Calcula el valor principal del arco seno de x. Puede producirse un error de dominio para los argumentos que no estén en el intervalo [-1, +1].
atan()	double atan(double x);	Calcula el valor principal del arco tangente de x.
atan2()	double atan2(double y, double x);	Calcula el valor principal del arco tangente de y/x, usando los signos de ambos argumentos para determinar el cuadrante del valor de retorno. Puede producirse un error de dominio si ambos argumentos son cero.
ceil()	double ceil(double x);	Calcula el valor integral más pequeño que no sea menor de x
cos()	double cos(double x);	Calcula el coseno de x (medido en radianes).
cosh()	double cosh(double x);	Calcula el coseno hiperbólico de x. Puede producirse un error de recorrido si la magnitud de x es demasiada grande.
exp()	double exp(double x);	Calcula la función exponencial de x.
fabs()	double fabs(double x);	Calcula el valor absoluto del número de coma flotante, x.
floor()	double floor(double x);	Calcula el valor integral más grande que no sea mayor de x.
Ffmod()	double fmod(double x, double y);	Calcula el resto de coma flotante de la división de x/y.
frexp()	double frexp(double valor, int *exp);	Parte en dos el número de coma flotante en una fracción normalizada y un entero con potencia a la 2. Guarda el entero en el objecto int apuntado por exp.

ldexp()	double ldexp(double x, int exp);	Multiplica un número de coma flotante y un entero con potencia a la 2. Puede producirse un error de recorrido
log()	double log(double x);	Calcula el logaritmo natural (o neperiano). Puede producirse un error de dominio si el argumento es negativo. Puede producirse un error de recorrido si el argumento es cero
log10()	double log10(double x);	Calcula el logaritmo en base 10 de x. Puede producirse un error de dominio si el argumento es negativo. Puede producirse un error de recorrido si el argumento es cero
pow()	double pow(double x, double y);	Calcula x elevado a la potencia de y. Puede producirse un error de dominio si x es negativo e y no es un valor entero. También se produce un error de dominio si el resultado no se puede representar cuando x es cero e y es menor o igual que cero. Un error de recorrido puede producirse.
sin()	double sin(double x);	Calcula el seno de x (medido en radianes).
sinh()	double sinh(double x);	Calcula el seno hiperbólico de x. Aparece un error de recorrido si la magnitud de x es demasiada grande
sqrt()	double sqrt(double x);	Calcula la raíz cuadrada del valor no negativo de x. Puede producirse un error de dominio si x es negativo.
tan()	double tan(double x);	Calcula la tangente de x (medido en radianes).
tanh()	double tanh(double x);	Calcula la tangente hiperbólica de x.

Procedimiento

1. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el siguiente ejemplo, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la practica en tu dispositivo de almacenamiento

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

main()
{
 double A=2.1, B=5.3, X=0.74, Y=1.01, Z=3.1;
 int N=3;
printf("sin(A) =  %6.4f\t",sin(A));   printf("cos(A)  =  %6.4f\n", cos(A));
printf("tan(A) =  %6.4f\t", tan(A));  printf("asin(X) =  %6.2f\n", asin(X));
printf("acos(A)=  %6.2f\t", acos(X)); printf("atan(Y) =  %6.2f\n", atan(Y));
printf("sinh(B)=  %6.4e\t", sinh(B)); printf("cosh(B) =  %6.4e\n", cosh(B));
printf("tanh(B)=  %6.4e\t", tanh(B)); printf("exp(B)  =  %6.4e\n", exp(B));
printf("log(B) =  %6.4e\t", log(B));  printf("log10(B)=  %6.4e\n", log10(B));
printf("pow(X,Z)= %6.4e\t", pow(X,Z));printf("sqrt(B) =  %6.4e\n", sqrt(B));
printf("ceil(Z)=  %3.1e\t", ceil(Z)); printf("floor(Z)=  %3.1e\n", floor(Z));
printf("fabs(B)=  %6.4e\t", fabs(B)); printf("ldexp(B,N)=%6.4e\n",
  ldexp(B,N));
printf("fmod(B,A)=%6.4e\n", fmod(B,A));
getch();
return 0;
 }
```

En el cuadro siguiente escribe una aplicación en la que se use la librería math

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificación del algoritmo			
Propuesta			

Tema 12: Uso de cadenas

Duración Estimada: 1 sesión de 50 minutos

Competencia 4: Usar arreglos unidimensionales para almacenar información

Objetivo: El alumno usará las estructuras básicas del manejo de cadenas de caracteres

Introducción Teórica:

Para manipular cadenas de caracteres se requiere de funciones como la función strcpy. Las funciones para manipular cadenas de caracteres se encuentran definidas en la librería string.h. Por lo tanto, va a ser necesario que incluyamos una nueva librería en nuestros programas con cadenas de caracteres: **#include <string.h>**

Comparación de Cadenas de Caracteres

Frecuentemente es necesario comparar cadenas de caracteres entre sí. Otra vez, en cadenas de caracteres se tiene una excepción respecto de lo que se ha visto. Para comparar dos cadenas de caracteres **no se pueden usar los operadores ==**, **>=** ó **<=**. Para comparar dos cadenas se utiliza una función especial (también definida en string.h) llamada **strcmp**. strcmp recibe dos argumentos, que son las dos cadenas a comparar, y regresa un valor de tipo entero. Por ejemplo, para las mismas declaraciones de nombre1 y nombre2 dadas anteriormente:

x = strcmp(nombre1, nombre2); es una sentencia correcta si x es de tipo entero. El valor que regresa la función strcmp es 0 si las dos cadenas son iguales. Regresa 1 si la primera cadena es mayor a la segunda y regresa –1 si la segunda cadena es mayor a la primera. Aquí, ser mayor no significa tener más caracteres, sino la comparación se hace considerando el número de código ascii de los caracteres. La comparación se hace uno a uno hasta que se encuentre un carácter diferente entre las dos cadenas.

Otras dos funciones para Cadenas de Caracteres

Existen otras dos funciones (aunque hay mucho más) que son de uso muy común para manipular cadenas de caracteres. Estas son las funciones **strlen** y **strcat**.

La función **strlen** recibe como argumento una cadena y da como valor de regreso un entero que corresponde al número de caracteres de la cadena (sin contar al carácter nulo). Por ejemplo, en el caso siguiente: char nombre[10]=”Juan”;

```
int x;        x = strlen(nombre);
```

La variable x tendría un valor de 4 luego que se ejecutan las sentencias.

La función **strcat** recibe como argumentos dos cadenas y da como resultado la unión de ambas cadenas en el orden indicado. La segunda cadena se anexa a la primera cadena. Por ejemplo, las siguientes sentencias:

char nombre[20]="Juan ", apellido[10]="Razo";

strcat(nombre,apellido); cambia el valor de la cadena *nombre* de "Juan " a "Juan Razo".

Para usar strlen y srtcat también se necesita string.h.

Uso de Cadenas con Funciones y Salidas de Resultados y Entradas de Datos

Debe destacarse que, cuando se trata de cadenas de caracteres, en las funciones strlen, strcat, strcmp y strcpy, se usa únicamente el nombre del arreglo que contiene a las cadenas, no se utiliza su dimensión. Esta es otra excepción a lo que se vio antes. Es decir, se usó, por ejemplo:

x = strlen(nombre); y no *x = strlen(nombre[20]);*

Arreglos de Variables para Almacenar Cadenas de Caracteres

Hemos visto que, para guardar una cadena que tenga como máximo 19 caracteres se utilizó:

char nombre[20];

¿Qué pasaría, sin embargo, si uno estuviera haciendo una base de datos de 10 nombres?. Una opción sería por supuesto usar:

char nombre1[20], nombre2[20], ..., nombre10[20]; Sin embargo, una opción más sencilla y eficiente es usar arreglos multidimensionales. Así, por ejemplo, si *char nombre[20];*

Se usó para un solo nombre de 19 caracteres (máximo), la siguiente sentencia se puede usar para definir una variable que pueda contener 10 nombres de 19 caracteres:

char nombre[10][20]; Así, cada uno de nombre[0], nombre[1], ..., nombre[9] podría almacenar una cadena de 19 caracteres.

Procedimiento:

Codifique el siguiente código y explique el trabajo de cada sentencia en su cuaderno

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main(){
//_____________________INICIALIZAR EL ARREGLO _________________
char coches[10][15]={"Alfa Romeo","Fiat","Ford", "Lancia", "Renault","Seat"};
int i,n=10,p,c;

//____MOSTRAR EL ESTADO ORIGINAL DEL ARREGLO_____________________
for(i=0;i<=9;i++){
  printf("Auto %d: %s \n",i,coches[i]);
```

```
}
getchar();

printf("Cambiando de posicion para insertar en posicion 4");getchar();
// calcular la posicion del elemento a insertar p(5)
p=4;
//Inicializar contador de insercion
i=n;
while(i>=p){
      //mover el elemento i-esimo hacia abajo a la posicion i+1
      for(c=0;c<15;c++)
      coches[i+1][c]=coches[i][c];
      //decrementar el contador
      i=i-1;
      }
//insertar el elemento en la posicion p

strcpy(coches[p], "Opel");
 //actualizar el contador de elementos del vector
n=n+1;
//________MOSTRAR EL ESTADO DEL ARREGLO_____________________________
for(i=0;i<=9;i++){
printf("Auto %d: %s \n",i,coches[i]);
}
getchar();
getchar();
}
```

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código del ejemplo ejecutándose	Entrega a tiempo	Calificación	Firma o sello
Codificación del ejemplo en máquina			
Explicación del código en cuaderno			

Tema 13: Uso de la estructura while( ) y for (; ;)

Duración Estimada: 1 sesión de 50 minutos

Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará la estructura repetitiva for () y while()

Introducción teórica:

Los procesos repetitivos son la base del uso de las computadoras. En estos procesos se necesita normalmente contar los sucesos, acciones o tareas internas del ciclo.

Una estructura cíclica o estructura repetitiva es aquella que le permite al programador repetir un conjunto o bloque de instrucciones un número determinado de veces mientras una condición dada sea cierta o hasta que una condición dada sea cierta. Se debe establecer un mecanismo para terminar las tareas repetitivas. Dicho mecanismo es un control que se evalúa cada vez que se realiza un ciclo. La condición que sirve de control puede ser verificada antes o después de ejecutarse el conjunto de instrucciones o sentencias. En caso de que la verificación o evaluación resulte verdadera se repite el ciclo o caso de ser falsa lo terminará.

Ciclos con control antes (For y While)

Las estructuras cíclicas cuyo control esta antes del ciclo, son estructuras que realizan la evaluación antes de ejecutar el bloque de instrucciones que tiene que repetir. Dependiendo de la evaluación que se realice se ejecutara o no dicho conjunto de instrucciones. Es posible que no se realice ni una sola vez ese conjunto de instrucciones.

while (*expresion* **)** *sentencia*

for (*expresion1* **;** *expresion2* **;** *expresion3* **)** *sentencia*

Procedimiento:

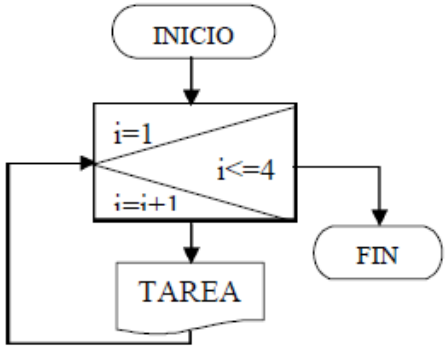
1. Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el siguiente ejemplo, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la practica en tu dispositivo de almacenamiento

```
  Ejemplo 1:
Este programa da una lista de los caracteres ASCII imprimibles, con código
numérico mayor a 32 (40 en octal) y menor que 127 (177 en octal).
*/
#include <stdio.h>
main()
{
 char c='\33', CR=0;
 printf("LISTA DE CARACTERES Y SUS VALORES ASCII\n");
 while (c++ <'\177')
      {printf("%c\t%d\t",c,c);getch();
      if(CR++ >= 4){printf("\n"); CR=0;}
```

```
    }
return 0;}
```

Ejemplo 2:

```
/* Programa que escribe los cuatro primeros
números */
#include "conio.h"
#include "stdio.h"
int i;
main()
{ clrscr();
for(i=1; i<=4; i++)
printf("%d ",i);
getch();
}
```



En el cuadro siguiente escribe el código necesario para que en la pantalla se visualice un marco semejante.

Código

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código de los ejemplos ejecutándose	Entrega a tiempo	Calificación	Firma o sello
Codificación de los ejemplos			
Propuesta			

Tema 14: Codificación de un ciclo do while( )

Duración Estimada: 1 sesión de 50 minutos

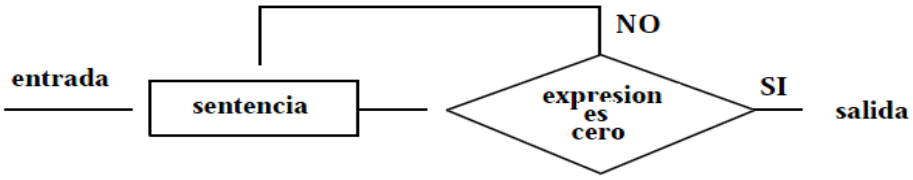
1. Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las reglas de sintaxis básicas del lenguaje C para codificar algoritmos en los que se aplique la estructura **do ... while()**

Introducción Teórica:

La sentencia " do " es una variante de "while". Tiene la finalidad de repetir un conjunto de sentencias. En "do", el test se hace al final del bucle.

do *sentencia* **while (** *expresión* **);**



Estructura Sencilla:

Forma general de uso:

Ejecuta bloque de instrucciones uno
Si (Condición) entonces
Repetir
Sino fin

Ejemplo:

```
/* Se repite la orden de lectura mientras el valor introducido sea menor o igual que cero */
{
int n;
do {
printf("Entero positivo ? ");
scanf("%d", &n);
} while ( n<=0 );
}
```

Procedimiento

1 Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el siguiente ejemplo, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la práctica en tu dispositivo de almacenamiento.

2 En el cuadro siguiente escribe el código necesario para mostrar un menú y que el programa termine cuando el usuario seleccione la opción salir

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificación del algoritmo			

Tema 15: Codificación de la estructura switch ()

Duración Estimada: 1 sesión de 50 minutos

Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno usará las reglas de sintaxis básicas del lenguaje C para codificar algoritmos en los que se aplique la estructura **switch ().**

Introducción teórica

La sentencia " switch " Es una sentencia condicional múltiple que generaliza a la sentencia "if else".

1. Se evalúa la "expresion_entera"
2. Se ejecuta la claúsula "case" que se corresponda con el valor obtenido en el paso
    1. Si no se encuentra correspondencia, se ejecuta el caso "default"; y si no lo hay, termina la ejecución de "switch".
3. Termina cuando se encuentra una sentencia "break" o por caída al vacío.

La estructura básica para usarla es la siguiente

switch (*expresion_entera*) {
case *expresion_constante_entera* : *sentencia*
break;
case *expresion_constante_entera* : *sentencia*
break;
. . .
default : *sentencia*
}

Ejemplo en lenguaje C

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
void main ( )  {
float r, rad;
float raiz(float (*f)(float), float);
float estandar(float), aprox(float);
char opcion;
printf("Introducir radicando y opcion(1/2)\n");
scanf("%f %c", &rad, &opcion);
switch ( opcion ) {
case '1' : r=raiz(estandar, rad); break;
case '2' : r=raiz(aprox, rad); break;
```

```
default : printf("opcion incorrecta\n"); exit(-1);
}
printf("radicando: %f opcion: %c raiz: %f\n",
rad, opcion, r);
}
float raiz( float (*f)(float), float v )
{ return( (*f)(v) ); }
float estandar(float rad)
{ return( sqrt(rad) ); }
float aprox(float rad)
{
float r1, r2=1.0;
do {
r1=r2;
r2=( rad/r1 + r1 )/(float)(2);
} while ( abs(r2-r1) > 1e-3*r1 ); return(r2);
}
```

Procedimiento

1 Con las instrucciones del profesor, usa el entorno de desarrollo para codificar el siguiente ejemplo, una vez que lo tengas capturado depúralo y ejecútalo para verlo funcionar. NO olvides guardar la práctica en tu dispositivo de almacenamiento.

2 En el cuadro siguiente escribe el código necesario para completar el programa de la practica anterior y lograr que además de mostrar un menú realice varias tareas; recuerda que el programa debe terminar cuando el usuario seleccione la opción salir

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Código ejecutándose de los ejemplos	Entrega a tiempo	Calificación	Firma o sello
Codificación del algoritmo			

Tema 16: Solución de problemas ()

Duración Estimada: 8 sesiones de 50 minutos

Competencia 3: Utilizar los elementos básicos de un lenguaje de programación estructurado para la codificación de algoritmos y / o diagramas de flujo

Objetivo: El alumno conocerá y usará las reglas para la construcción de algoritmos, los codificará, y los implementará en otras máquinas

Introducción teórica

Desde que las ideas de Knuth, Dijkstra y Wirth fueron consolidadas en el campo informático, las reglas para la construcción de algoritmos han ido variando constantemente y de igual forma los lenguajes de programación, en general se han ido adaptando a estas reglas o técnicas de programación.

Las reglas que se deben considerar en una buena programación son:

1. Diseñar algoritmos en etapas yendo de lo general a lo particular (método descendente)
2. Dividir el algoritmo en partes independientes -módulos- y tratar cada módulo independientemente.
3. Establecer y utilizar la solución de problemas con técnicas de programación estructurada
4. Dar especial importancia a las estructuras de datos
5. Describir completamente cada algoritmo
6. Verificar o realizar la prueba de escritorio a cada algoritmo desarrollado.

Un programa puede ser considerado como el conjunto de **Estructuras de datos** más el conjunto de **Estructuras de programación** más el **Encabezado** que exige cada lenguaje en particular.

PROGRAMACIÓN ESTRUCTURADA

La programación estructurada es el conjunto de técnicas para desarrollar programas fáciles de escribir, verificar, leer y mantener. Se puede concretar más la definición diciendo que la programación estructurada es el conjunto de técnicas que incluye:

Un número limitado de estructuras de programación

- Diseño descendente

- Descomposición modular con independencia de los módulos

El teorema de Bohm y Jacopini establece que un programa propio puede ser escrito utilizando solo tres tipos de estructuras de control:

- Secuencial
- Selectiva
- Repetitiva

Hoy me atrevería a decir que un programa propio puede ser escrito utilizando lo siguiente:

Declaraciones:

- Librerías de inclusión
- Declaración de funciones y/o procedimientos
- Definición de constantes y/o variables

Estructuras de programación:

- Asignación
- Decisión
- Cíclicas
- De selección múltiple

Estructuras de datos:

- Estáticas simples
- Dinámicas
- Registros
- Arreglos
- Archivos
- Funciones:
- Predefinidas por el lenguaje
- Definidas por el usuario.

Generalmente cuando el algoritmo o solución de un problema determinado se deja en términos de diagrama de flujo, pseudolenguaje e incluso en pseudocódigo se puede trabajar únicamente con estructuras de programación.

Procedimiento

Con las instrucciones del profesor, analiza por lo menos 10 de los siguientes problemas, diseña el algoritmo, codifica la solución y ejecútalos en una máquina diferente.

Problemas

1. Un alumno desea saber cuál será su calificación final en la materia de algoritmos, dicha calificación se compone de los siguientes porcentajes, 55% del promedio de sus tres calificaciones parciales, 30% de la calificación del examen
2. Una tienda ofrece un descuento del 15% sobre el total de la compra, y un cliente desea saber cuánto deberá pagar finalmente por su compra

3. Un vendedor recibe un sueldo base más un 10 % extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes, y el total que recibirá en el mes
4. Escribir un programa que permita determinar el mayor de tres números enteros distintos.
5. Escribir una programa que permita ordenar de menor a mayor tres números enteros distintos.
6. Escribir un programa que permita efectuar el cálculo simplificado de impuestos de acuerdo con el siguiente esquema:
    a. Si el monto total es menor de $50.000, entonces calcular Tasa=0,16 * Monto;
    b. Si el monto total es mayor o igual a $50.000 y menor que $100.000 entonces calcular Tasa=0,16* Monto+0,2* (Monto-50.000);
    c. Si el monto total es mayor o igual que $100.000 entonces calcular Tasa=0,16* Monto+0,2* (Monto-50.000)+0,25* (Monto-100.000).
7. Escribir un programa que calcule las raíces de una ecuación de segundo grado del tipo $ax^2+bx+c=0$.
8. Escribir un programa que permita transformar un número entero de tres dígitos en el correspondiente número romano.
9. Dadas las longitudes de tres segmentos A, B y C, escribir un programa que decida si con éstos se puede formar un triángulo equilátero o un triángulo isósceles o un triángulo escaleno o ninguno de los anteriores.
10. Escribir un programa que permita determinar si un punto se encuentra en el interior de un rectángulo, o en alguno de los lados o en el exterior, teniendo como datos dos vértices opuestos de dicho rectángulo.
11. Escriba un programa que permita tabular la equivalencia entre *ºF* y *ºC*, ingresando como dato *ºC* y utilizando la fórmula de conversión *ºF = 9/5 ºC + 32*, en un intervalo que va de un valor *A* hasta un valor *B* con incrementos iguales a *H*.
12. Modifique la programa anterior para tabular *y* = sen(*x*), considerando que los valores de *A*, *B* y *H* se ingresan en grados.
13. Escriba un programa que simule tirar un dado. Modifíquelo para que cuente la cantidad de veces que se debe tirar un dado hasta que salga un número entrado por el usuario.
14. Escriba una programa que permita calcular un valor aproximado para *x* tal que se cumpla que *x = cos(x)*.
15. Escriba un programa que calcule el promedio de las *N* notas de los exámenes de cada alumno para un curso de *M* alumnos.
16. Modifique el programa obtenido en el ejercicio anterior para repetir el experimento 1000 veces y calcular el promedio obtenido en los 1000 experimentos.
17. Escriba un programa que permita determinar si un número entero *Z* es primo o no lo es. Hacer uso del teorema:
    Para todo número primo p > 3, se tiene que p=6k+1 ó p=6k-1
    Demostración: Todos los enteros pueden expresarse exactamente de un de las 6 posibles formas: 6k, 6k+1, 6k+2, 6k+3, 6k-2, ó 6k-1

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Diseño del algoritmo	Código ejecutándose en máquina diferente	Entrega a tiempo	Firma o sello	Calificación
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.				

Evidencia: Archivos en el dispositivo de almacenamiento y ejecución en máquinas diferentes

Tema 17: Arreglos Unidimensionales y Bidimensionales ()

Duración Estimada: 4 sesiones de 50 minutos

Competencia 4: Utilizar arreglos unidimensionales y bidimensionales para el almacenamiento de datos en la solución de problemas

Objetivo: El alumno conocerá y usará los arreglos unidimensionales y bidimensionales para almacenar los datos necesarios para la solución de problemas mediante programas computacionales.

Introducción teórica:

Arreglos unidimensionales y multidimensionales

Los arreglos son una colección de variables del mismo tipo que se referencian utilizando un nombre común. Un arreglo consta de posiciones de memoria contigua. La dirección más baja corresponde al primer elemento y la más alta al último. Un arreglo puede tener una o varias dimensiones. Para acceder a un elemento en particular de un arreglo se usa un índice. El formato para declarar un arreglo unidimensional es:

```
tipo nombre_arr [ tamaño ]
```

Por ejemplo, para declarar un arreglo de enteros llamado **a** con diez elementos se hace de la siguiente forma:

```
int a[10];
```

En C, todos los arreglos usan cero como índice para el primer elemento. Por tanto, el ejemplo anterior declara un arreglo de enteros con diez elementos desde **a[0]** hasta **a[9]**.

La forma como pueden ser accesados los elementos de un arreglo, es de la siguiente forma:

```
a[2] = 15; /* Asigna 15 al 3er elemento del arreglo a*/
num = a[2]; /* Asigna el contenido del 3er elemento a la variable num */
```

El lenguaje C no realiza comprobación de contornos en los arreglos. En el caso de que sobrepase el final durante una operación de asignación, entonces se asignarán valores a otra variable o a un trozo del código, esto es, si se dimensiona un arreglo de tamaño ***N***, se puede referenciar el arreglo por encima de ***N*** sin provocar ningún mensaje de error en tiempo de compilación o ejecución, incluso aunque probablemente se provoque un error en el programa.

Como programador se es responsable de asegurar que todos los arreglos sean lo suficientemente grandes para guardar lo que pondrá en ellos el programa.

C permite arreglos con más de una dimensión, el formato general es:

```
tipo nombre_arr [ tam1 ][ tam2 ] ... [ tamN];
```

Por ejemplo un arreglo de enteros bidimensionales se escribirá como: `int b[50][50];`

Observar que para declarar cada dimensión lleva sus propios paréntesis cuadrados.

Para acceder los elementos se procede de forma similar al ejemplo del arreglo unidimensional, esto es:

```
b[2][3] = 15; /* Asigna 15 al elemento de la 3ª fila y la 4ª columna*/
num = b[25][16];
```

A continuación se muestra un ejemplo que asigna al primer elemento de un arreglo bidimensional cero, al siguiente 1, y así sucesivamente.

```
main() {
int t,i,num[3][4];
for(t=0; t<3; ++t)
 for(i=0; i<4; ++i)
 num[t][i]=(t*4)+i*1;
for(t=0; t<3; ++t){
 for(i=0; i<4; ++i)
 printf("num[%d][%d]=%d ", t,i,num[t][i]);
 printf("\n");
}
}
```

En C se permite la inicialización de arreglos, debiendo seguir el siguiente formato:

```
tipo nombre_arr[ tam1 ][ tam2 ] ... [ tamN] = {lista-valores};
```

Por ejemplo:

```
int c[10] = {1,2,3,4,5,6,7,8,9,10};
int num[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
```

Procedimiento:

Codifica los ejemplos siguientes y explica línea a línea la semántica de cada instrucción, posteriormente resuelve los siguientes problemas y muestra los resultados a tu profesor:

```
/********************** Programa ARREGLOS.C ***************************
  Este programa ilustra el manejo de ARREGLOS unidimensionales y
  bidimensionales
*/
#include <stdio.h>
main()
{
 int VECTOR1[10] = {0, 10, 20, 30, 40, 50, 60, 70, 80, 90}; //arreglo unidimensional
 int MATRIZ_A[3][4] = {0, 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23};//arreglo bidimensional
 int fila, columna;
  printf("\n\n\n\n\n\n\n\n\n\n");
  printf("ELEMENTOS DE VECTOR1: \n");
  for (fila=0; fila<10; fila++)  {
   printf("VECTOR1[%d] =  %d\n", fila, VECTOR1[fila]);  }
  printf("\nELEMENTOS DE MATRIZ_A: \n");
  for (fila=0; fila<3; fila++)
   for (columna=0; columna<4; colu  mna++)    {
     printf("MATRIZ_A1[%d][%d] =  %d\n", fila,columna,MATRIZ_A[fila][columna]);
    }
    getchar();   }
```

Lista de problemas:

1. Haga un programa que use tres arreglos en ellos se almacenará la siguiente información:
   a. En uno necesitamos guardar 15 nombres de alumnos

b. En el segundo arreglos almacenaremos las calificaciones de 5 materias del cada uno de los alumnos
c. En el tercero almacenaremos el promedio de cada alumno

Nombre de Alumno	Calificaciones					Promedio de calificaciones
Luis Méndez Huerta	8.8	9.2	6.7	7.9	6.9	7.9

2. Haga un programa que use tres arreglos, en ellos se debe almacenar, en el primero el nombre de 45 empleados en el segundo se debe guardar el departamento en el que trabajan en el tercero se almacenara las ventas, su comisión y su sueldo.
   a. El programa deberá incluir la introducción de datos y la muestra de los datos en forma de tabla:

Nombre de E	Departamento	Nomina		
Ana Pérez	Zapotlán 45 Texcoco	8500	2.125	5.125

3. Haga un programa que use tres arreglos de tipo flotante, en ellos se debe almacenar, en el primero el radio de 50 círculos, en el segundo se debe guardar el área de los 50 círculos y en el tercero se almacenara el volumen de 50 cilindros considerando que las alturas serán generadas de forma aleatoria en un rango de 1 a 20 cm.

Radio de Circulo	Área de Círculo	Volumen de cilindro
6.5	45.8	448.78

4. Haga un programa que use tres arreglos, en ellos se debe almacenar, en el primero el nombre de 45 empleados en el segundo se debe guardar el departamento en el que trabajan en el tercero se almacenara las ventas, su comisión y su sueldo.
   b. El programa deberá incluir la introducción de datos y la muestra de los datos en forma de tabla:

Nombre de E	Departamento	Nomina		
Ana Pérez	Zapotlan 45 Texcoco	25	2.5	2000

Evaluación:

Se evaluará el desempeño del alumno en la ejecución de la práctica y los criterios serán los siguientes

Diseño del algoritmo	Código ejecutándose en máquina diferente	Entrega a tiempo	Firma o sello	Calificación
1.				
2.				
3.				
4.				

Anexos

El código ASCII define una relación entre caracteres específicos y secuencias de bits; además de reservar unos cuantos códigos de control para el procesador de textos, y no define ningún mecanismo para describir la estructura o la apariencia del texto en un documento.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Actividades Complementarias

Evaluación Inicial

Objetivo: Identificar los conocimientos previos del alumno y su experiencia en la programación, así como sus expectativas del curso.

Instrucciones: Responde las preguntas que se te hacen a continuación y entrégalo a tu profesor.

Nombre:_____

Sección 1:

1. ¿Qué es un programador? _____
2. ¿Qué es un lenguaje de programación? _____
3. ¿Qué es un compilador? _____
4. ¿Qué es el lenguaje máquina? _____
5. ¿Qué es un algoritmo? _____
6. ¿Has codificado algún algoritmo, en que lenguaje? _____
7. ¿Qué es la programación estructurada? _____
8. ¿Qué es un ciclo? _____
9. ¿Qué es una instrucción secuencial?_____
10. ¿Qué es el programa fuente? _____

Sección 2: Define con tus palabras los siguientes conceptos:

Computadora

Memoria RAM

Entorno de Programación

CPU

Datos de entrada

Datos de salida

Análisis del problema

Diseño de algoritmo

Codificación de algoritmo

Programa objeto

Archivo

Lenguaje de programación

Sección 3

1. ¿Por qué te interesa la Ingeniería en Computación?

2. ¿Por qué crees que es útil la programación?

3. ¿Por qué crees que se necesitan los programas de cómputo?

4. ¿Qué importancia tiene la programación en la carrera de Ingeniería en Computación?

5. ¿Qué tipo de programas te gustaría aprender a codificar?

6. Si ya has programado ¿Qué tipo de programas has realizado?

Cuestionario

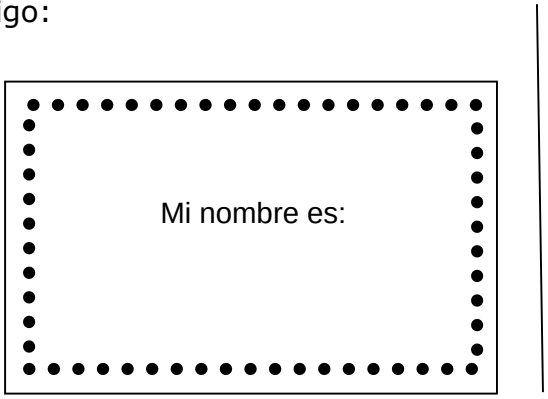
INSTRUCCIONES: Coloca en el paréntesis de la columna Preguntas la letra que corresponda de la columna Respuestas

Preguntas	**Respuestas**
( ) secuencia de escape que permite usar un tabulador horizontal en una sentencia pintf ( ) Configura el color del fondo de la pantalla. ( ) función usada para mostrar información en la pantalla ( ) Imprime un carácter en la sentencia printf. ( ) librería para usar entradas y salidas con formato ( ) función que permite limpiar de datos la pantalla ( ) configura el color del texto. ( ) imprime un valor entero en la sentencia printf ( ) permite usar un texto como comentario	1. printf( ); 2. \n 3. \t 4. \a 5. //texto 6. %s 7. %d 8. %f 9. %c 10. textcolor(1); 11. textbackground(8); 12. clrscr(); 13. conio.c 14. stdlib.h

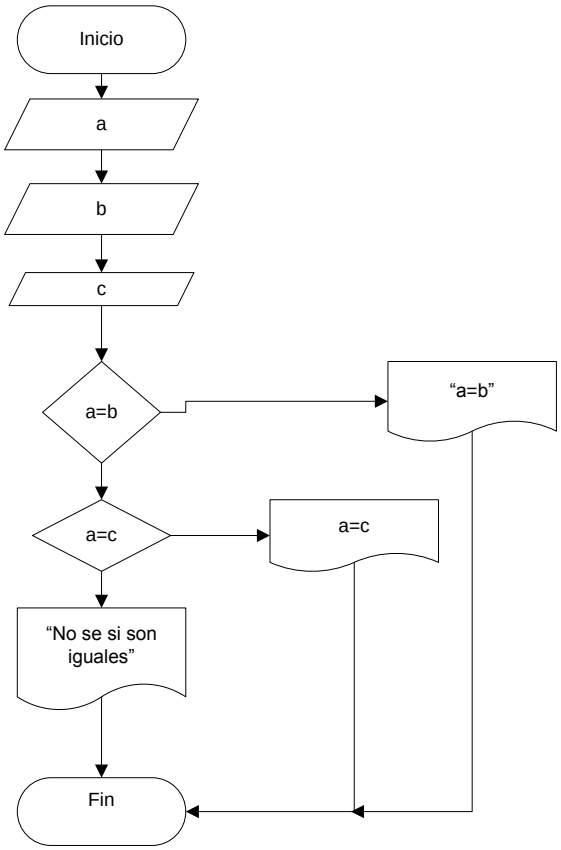
( ) librería que incluye funciones para modificar propiedades de pantalla(color, fondo, limpiar)	

Codifique el programa necesario para obtener en la pantalla la siguiente salida (use sentencias for)

Código:



Codifique el siguiente diagrama de flujo



Codifique un ejemplo de aplicación de las siguientes sentencias

for

if else

while

switch

do while

Ensayo de Autoevaluación

Instrucciones:

Realiza un ensayo con las siguientes características:

Tema: **Mis experiencias en la aplicación de la Programación estructurada**.

Extensión: Mínimo 5 cuartillas

Requisitos: Describir las experiencias vividas a través de la solución del manual, enfatizando en los siguientes aspectos:

- las dificultades que enfrentó
- hacer referencia a cada actividad de su portafolio resaltando el nivel de dificultad que percibió en cada una de ellas
- las fuentes bibliográficas que más le apoyaron
- el apoyo que logró en su trabajo colaborativo
- sus logros más satisfactorios

Valor de cada aspecto 2 puntos
Valor del ensayo 10

El ensayo final se integrará al portafolio de evidencias y permite que el alumno reflexione sobre los logros adquiridos durante el curso, y su proceso de aprendizaje permitiéndole identificar sus fortalezas y debilidades. La evaluación es continua y se define un valor numérico a partir de la integración del portafolio de evidencias que se formará con las actividades del curso.

Bibliografía

1. SZnajdleder Pablo Augusto, "Algoritmos a fondo con implementaciones en C y Java" AlfaOmega, Buenos Aires 2012

2. Cairó Osvaldo, "Fundamentos de Programación Piensa en C", Prentice Hall, México 2006

3. Cevallos Fco. Javier, "Enciclopedia de C", Editorial. Addison Wesley.

4. Deitel H. M. / Deitel P. J., "Como programar en C /C++". Editorial Prentice Hall Hispanoamericana.

5. Joyanes Aguilar Luís y Salonero Martínez Ignacio. "Programación en C", Editorial McGraw-Hill.

6. Tremblay Jean-Paul, Bunt Richard, "Introducción a la Ciencia de las computadoras enfoque", Mc Graw Hill, México 1988.

7. Vazquez Peña Mario "Introducción al lenguaje C", Universidad Autónoma Chapingo, México 1997