UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM TEXCOCO

INGENIERÍA EN COMPUTACIÓN

Problemario de Programación Estructurada

Septiembre 2018

Contenido

Presentación

Este compendio de ejercicios fue generado con la finalidad de apoyar a los alumnos a fortalecer las competencias que el alumno va desarrollando en las aulas. Puede usarse como problemario de la Programación Estructurada. Se omiten los textos extensos para que el alumno y el profesor puedan usarlo como banco de ejercicios para fortalecer lo ya visto en aula. S espera que ayude a los docentes en el proceso de la enseñanza de la programación y que sea una guía para los alumnos en la valoración de su aprendizaje.

Programación Estructurada es la primera Unidad de aprendizaje del área de docencia de Programación e Ingeniería del software del plan de estudios de Ingeniería en Computación de la Universidad Autónoma del Estado de México, es una unidad de aprendizaje elemental para la formación de un ingeniero en computación pues en ella se desarrollan las competencias profesionales básicas referentes al desarrollo de software que permitirán al ingeniero resolver problemas mediante programas de computación.

Esta Unidad de aprendizaje está seriada con Programación avanzada y es indispensable para cursar Estructura de datos. Las habilidades adquiridas en esta unidad de aprendizaje permiten que el alumno desarrolle su lógica de programación, la cual se irá incrementando o madurando al aprender diferentes paradigmas de programación durante la licenciatura. El programa de estudio de Programación Estructurada establece que debe impartirse como curso durante 6 horas a la semana; 3 horas prácticas y 3 horas teóricas y que corresponde al núcleo básico. En el mapa curricular se ubica al inicio del área disciplinar por lo que puede comprenderse la importancia de obtener buenos resultados en ella como base para desarrollar nuevas habilidades en unidades de aprendizaje subsecuentes.

El objetivo general del curso de acuerdo al plan de estudios es:

Aplicar el paradigma de la programación estructurada para representar en términos algorítmicos la solución de un problema real automatizable y elaborar programas completos utilizando el paradigma de la programación estructurada y mostrando en ellos el dominio pleno de variables simples, vectores, matrices, registros y el modularidad.

La estructura de la unidad de aprendizaje se compone de 5 unidades de competencias las cuales son:

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 arreglos unidimensionales y bidimensionales para el almacenamiento de datos en la solución de problemas.
4. Usar técnicas de programación modular en el desarrollo de programas informáticos
5. Utilizar los registros para almacenar y manipular información en el desarrollo de programas informáticos.

PROGRAMA DE ESTUDIO POR COMPETENCIAS

PROGRAMACIÓN ESTRUCTURADA

I. IDENTIFICACIÓN DEL CURSO

ESPACIO EDUCATIVO: Facultad de Ingeniería						
LICENCIATURA: Ingeniería en Computación **AÑO DE APROBACIÓN POR EL CONSEJO UNIVERSITARIO:**				**ÁREA DE DOCENCIA:** Programación e Ingeniería de Software		
APROBACIÓN POR LOS H.H. CONSEJOS ACADÉMICO Y DE GOBIERNO		**FECHA:**		**PROGRAMA ACTUALIZADO POR:** M. en I. Mireya Salgado Gallegos M. en A. Edith Salazar Vázquez M. en A. Silvia Edith Albarrán Trujillo M. en C. Eduardo Trujillo Flores Ing. Álvaro Arzate Trejo Ing. Leticia Hernández Linares		**PROGRAMA REVISADO POR:** Integrantes de la Academia de Programación e Ingeniería de Software
				FECHA DE ELABORACIÓN : Mayo 2007		**FECHA DE REVISIÓN :** Noviembre 2013
CLAVE	**HORAS DE TEORÍA**	**HORAS DE PRÁCTICA**	**TOTAL DE HORAS**	**CRÉDITOS**	**TIPO DE CURSO**	**NÚCLEO DE FORMACIÓN**
L41012	3	3	6	9	Curso-Laboratorio	Sustantivo
UNIDAD DE APRENDIZAJE ANTECEDENTE Ninguna				**UNIDAD DE APRENDIZAJE CONSECUENTE** Programación avanzada		
PROGRAMAS EDUCATIVOS O ESPACIOS ACADÉMICOS EN LOS QUE SE IMPARTE: Licenciatura en Ingeniería en Computación (Facultad. de Ingeniería, Centros Universitarios: Atlacomulco, Ecatepec, Texcoco, Valle de Chalco,						

Universidad Autónoma del Estado de México
UAEM

Secretaría de Docencia
Dirección de Estudios Profesionales

VII. ESCENARIOS DE APRENDIZAJE

Salón de clases

VIII. ESTRUCTURA DE LA UNIDAD DE APRENDIZAJE

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 diagramas de flujo y pseudocódigo. 3. Utilizar arreglos unidimensionales y bidimensionales para el almacenamiento de datos en la solución de problemas. 4. Usar las técnicas de programación modular en el desarrollo de programas informáticos. 5. Utilizar los registros para almacenar y manipular información en el desarrollo de programas informáticos.

Capítulo 1
La Programación y la solución de problemas

OBJETIVO: El alumno conocerá y aplicará las reglas para especificar algoritmos usando Diagramas de Flujo y/o pseudocódigo

1.1 LA PROGRAMACIÓN Y SU CONTEXTO

La Programación algorítmica es una actividad profesional que tiene como objetivo diseñar, codificar, depurar y mantener el código fuente de programas informáticos.

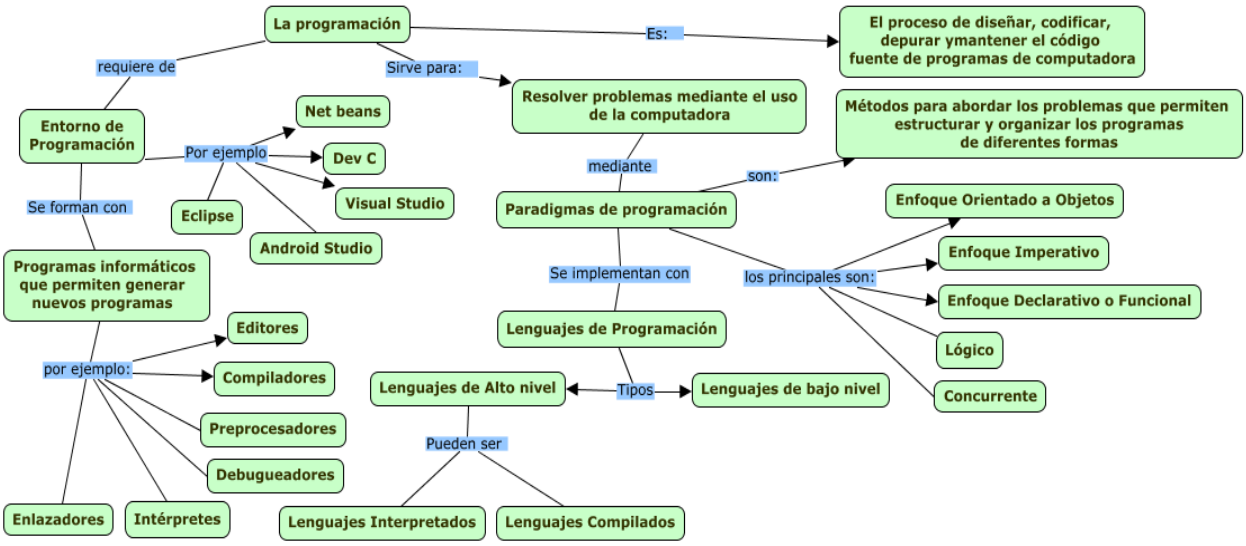


Imagen 1. Mapa conceptual sobre la programación y los lenguajes de programación.

1.2 Cuestionarios y ejercicios

1. ¿Qué hace el compilador?
2. ¿Cuáles son los pasos en el ciclo de desarrollo en el programa?
3. ¿Qué comando se necesita teclear para compilar un programa llamado PROGRAM1 en su compilador?
4. ¿Su compilador ejecuta el enlazado y la compilación con un sólo comando o se tiene que dar comandos separados?
5. ¿Qué extinción se debe usar para los archivos fuente del C?
6. ¿es FILENAME.TXT un nombre valido para un archivo fuente del C?
7. Explique con sus propias palabras el significado de la imagen 2

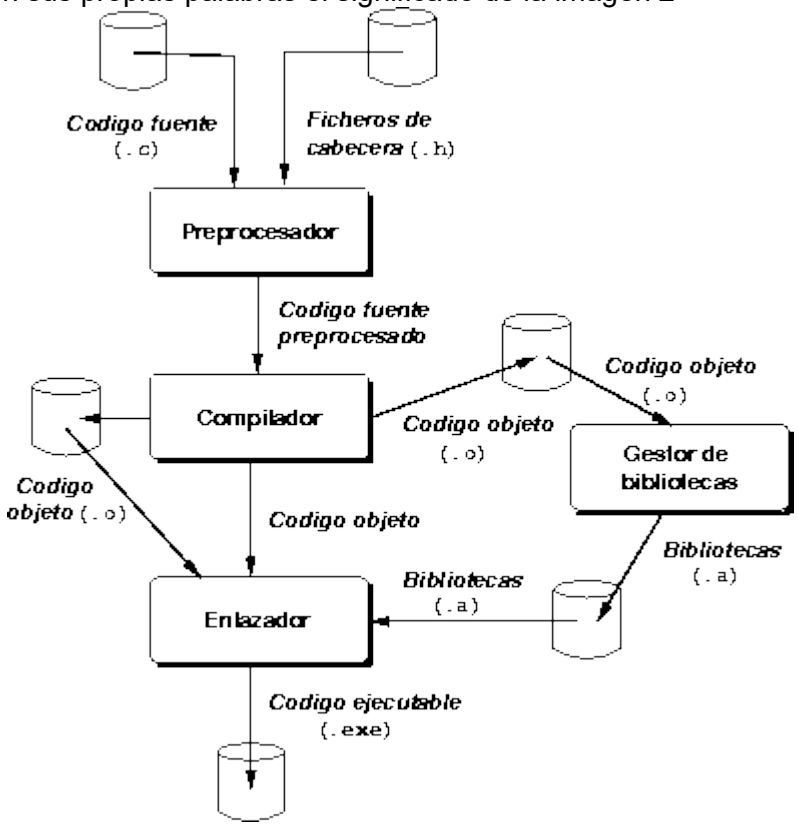


Imagen 2. Entorno de Compilación en C

8. Explique con sus propias palabras el significado de la imagen 3

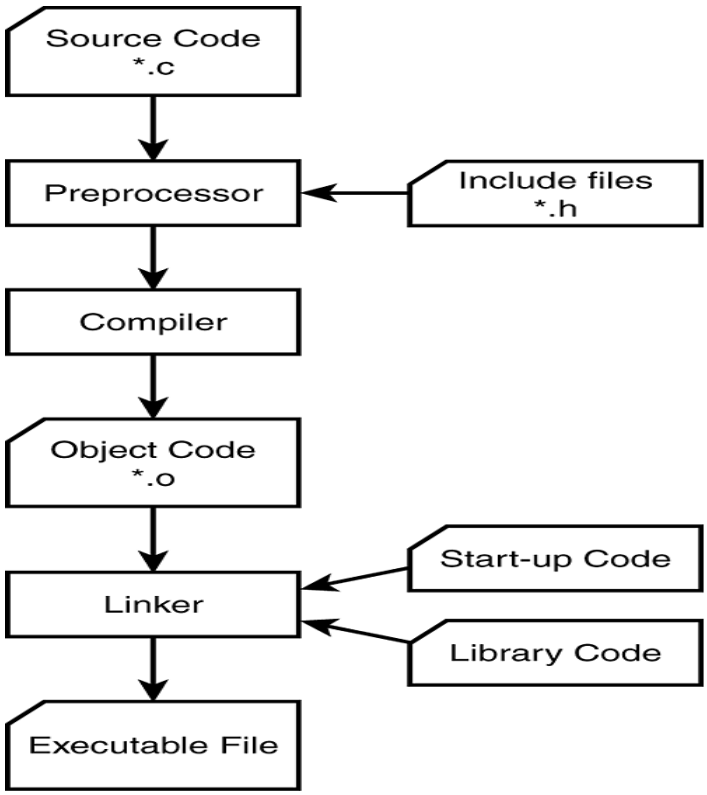


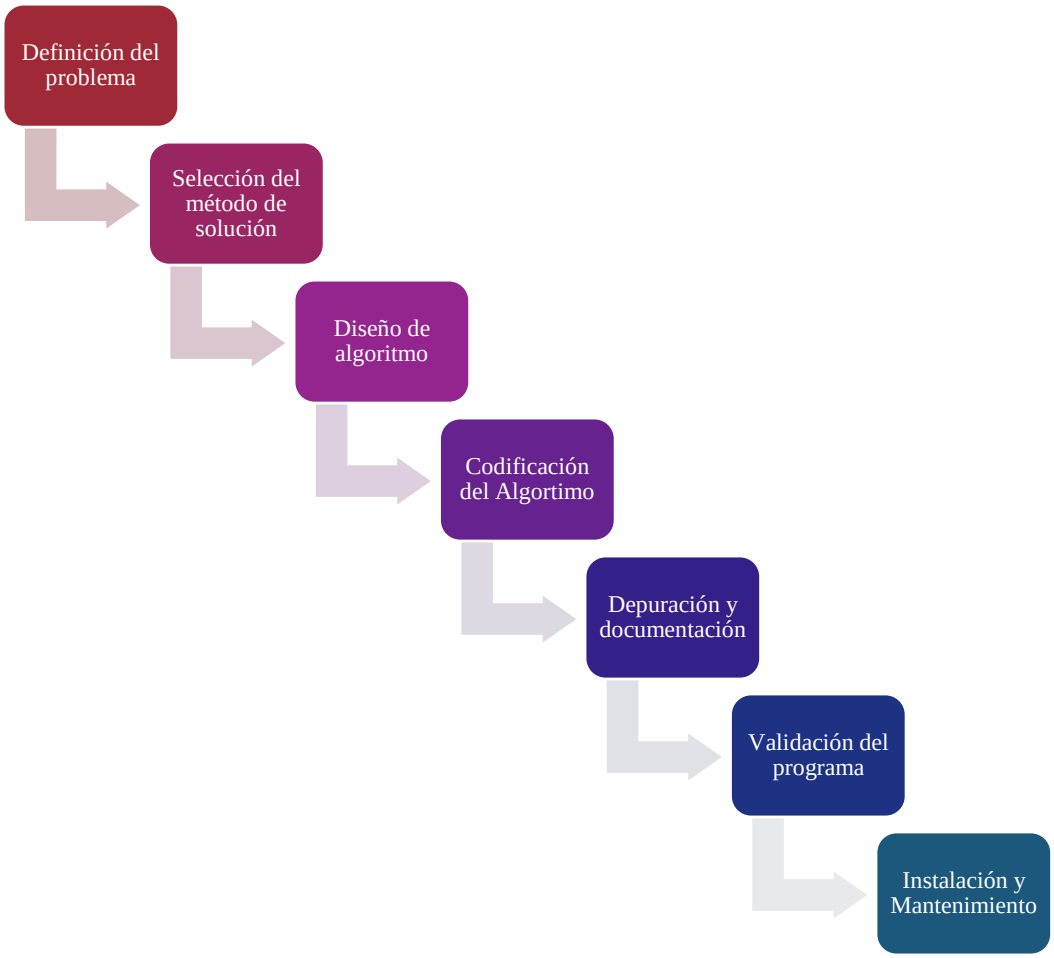
Imagen 3.

1.3 Fases en la solución de problemas

Para solucionar un problema con el apoyo de la computadora es necesario cumplir ciertas fases que deben ser secuenciales, el cumplimiento de ellas permite que el programador resuelva los problemas de forma sistemática y eficiente. El número de fases de la solución de problemas usando la computadora varía según los autores, pero en general se coincide en las siguientes:

1. Definir el problema: Es importante especificar con detalle y precisión, si no se conocen los detalles del problema difícilmente se podrán abordar con eficiencia. La definición de la problemática se logra mediante la recopilación de datos por medio de entrevistas, revisión de documentos y otras técnicas de comunicación y observación.
2. Selección del método de solución: con la definición del problema se tienen las bases necesarias para identificar un método de solución, esto puede requerir la valoración de varias soluciones
3. Diseño de algoritmo: los algoritmos son una técnica muy eficiente para representar los pasos ordenados en los que consiste la solución que pretendemos implementar para solucionar el problema

4. Codificación del algoritmo: en esta fase se implementan los pasos especificados en el algoritmo mediante un programa fuente realizado en algún lenguaje de programación
5. Depuración y documentación del programa: en este paso se depura el programa (corregir errores) y se documenta para tener un programa funcional y útil para ser entregado como solución
6. Validación de la solución: las personas que requieren la solución hacen pruebas del programa para verificar que es la solución que se necesitaba.
7. Instalación, producción y mantenimiento del programa: en esta fase se instala el programa y se usa en el entorno de operación necesario, se reproduce si fuera necesario y se le da mantenimiento, el mantenimiento implica agregar funcionalidad o adaptar rutinas para que el programa siga siendo útil



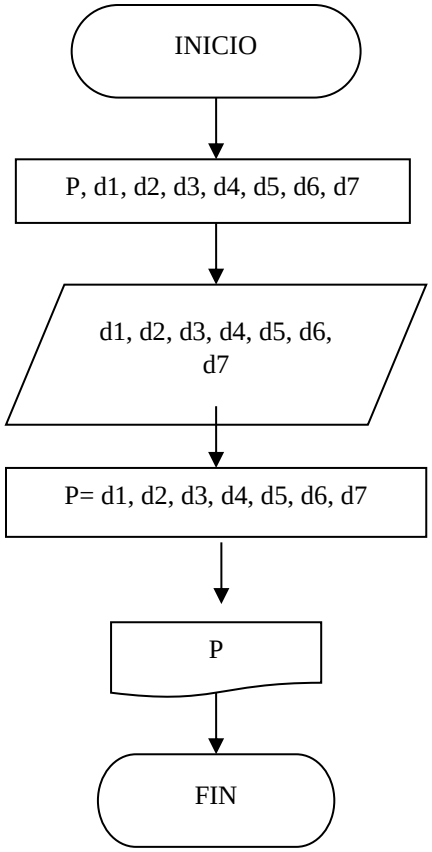
Antes de codificar es útil aprender a transitar por las primeras tres fases, pues el análisis del problema, la selección de una solución y su representación como un

algoritmo son fases clave para desarrollar programas útiles y eficientes. En el ejemplo siguiente se ejemplifica paso a paso cómo se resuelve un problema de naturaleza cotidiano.

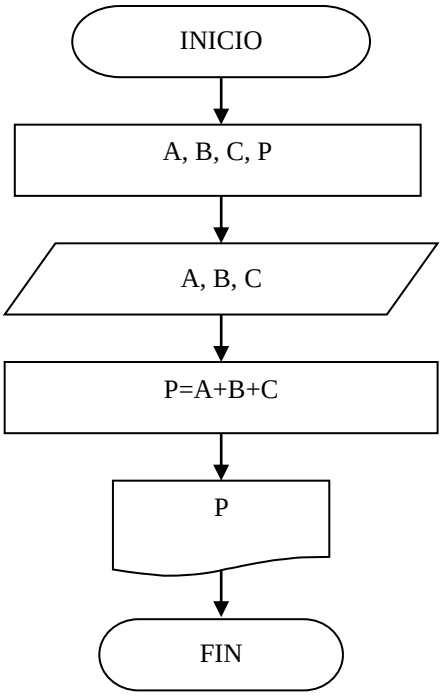
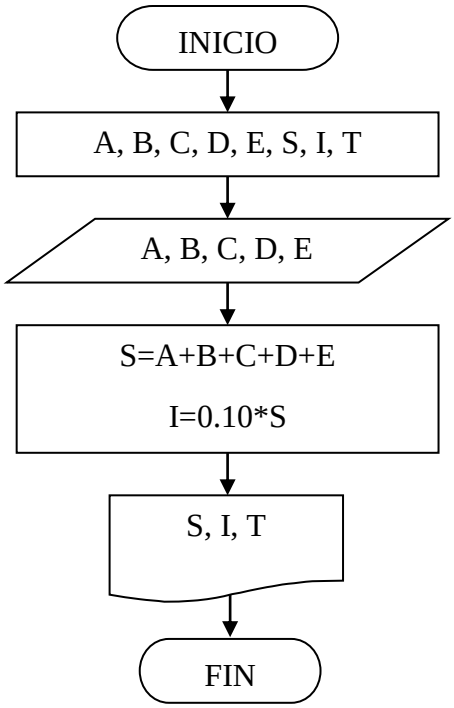
Ejemplo 1: Identificación de Algoritmos en la vida cotidiana

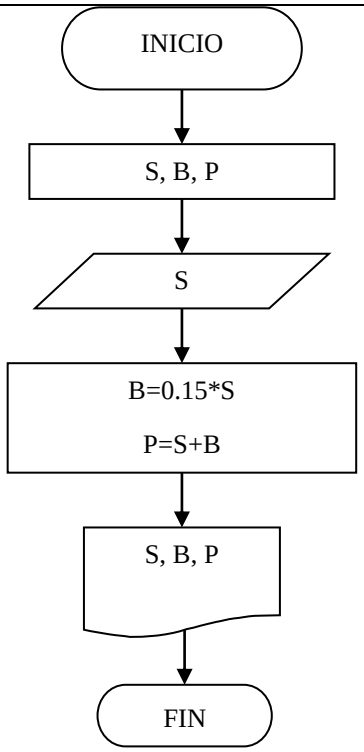
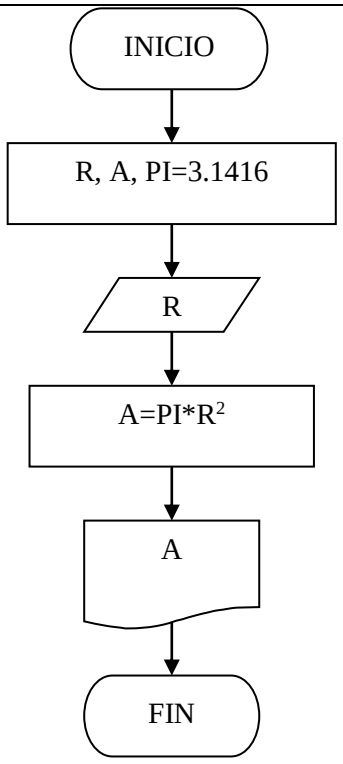
1: Plantear el programa claramente	Calcular el **promedio** de dinero gastado diariamente por una familia.	
1.1 Describir la información de entrada y de salida.	**Entrada:** Dinero gastado cada día durante una semana por la familia. **Salida:** Dinero promedio gastado por día.	
1.2 Resolver el problema a mano (o con una calculadora) para un conjunto de datos sencillo.	**Día**	**Dinero gastado**
	1	85
	2	164
	3	95.5
	4	76.5
	5	97
	6	103
	7	217
	Promedio:	(85+164+95.5+76.5+97+103+217)/7= 119.71
2. Solución	$D_{prom} = \frac{\sum_{n=1}^{i} n_1 + n_2 + \cdots + n_i}{i}$	
2.1. Probar el programa con diversos datos.	(257+92.5+261+120+66.5+77.6+85.5)/7=137.15	

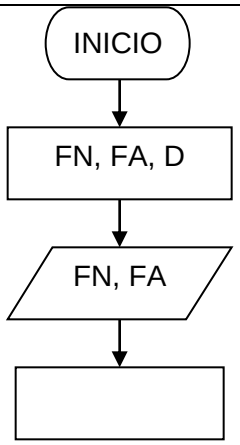
3: Diagramación del Algoritmo Computacional



Ejercicios: Realice los algoritmos que soluciones los problemas que se le plantean o explique lo que resuelve el algoritmo que se le presente

1. Se necesita un programa que pueda leer los 3 lados de un triángulo 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	

1.4 Cuestionarios y ejercicios

1. Se requiere conocer el área de un triángulo y sólo se conoce la longitud de sus tres lados, se sabe que el área de un triángulo cuyos lados son a, b, y c se puede calcular por la fórmula:
   $A = \sqrt{p \times (p-a) \times (p-b) \times (p-c)}$ Donde $p = (a+b+c)/2$. Proponga la solución que permitirá calcular el área de varios triángulos en las mismas condiciones.

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

2. Analice un programa que como resultado permita obtener la calificación total de un alumno con base en las calificaciones de 3 actividades (pr,ex,ta) con un porcentaje de 55% para prácticas, examen con un valor de 30%, y tareas 15%.

Análisis	**Pasos a seguir para resolver (Puede hacer un algoritmo)**
Datos de entrada: **Proceso:** **Datos de salida:**	

3. Analizar un programa que lea un número entero, lo multiplique por 2 y a continuación lo muestre en pantalla.

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

4. Analizar un programa que lea e imprima una serie de números distintos de cero. La función debe terminar con un valor cero que no se debe imprimir. Visualizar el número de valores leídos.

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

5. Analizar un programa para calcular la velocidad (en m/s) de los corredores de la carrera de 1500 metros. La entrada consistirá en parejas de números (minutos, segundos) que dan el tiempo del corredor; por cada corredor. El programa debe imprimir el tiempo en minutos y segundos y la velocidad. Ejemplo de entrada de datos: (3,53) (3,40) (3,46) (3,52) (0,0); el último par de datos se utilizará como fin de entrada de datos.

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

6. Analizar un programa que determine si un número N es primo. Recordamos que un número primo sólo se puede dividir por el mismo y por la unidad.

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

7. Escribir una función que determine si una palabra leída es capicúa. Ejemplo: radar

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

8. Escribir una función que cuente el número de ocurrencias de cada letra en una palabra leída como entrada.

Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

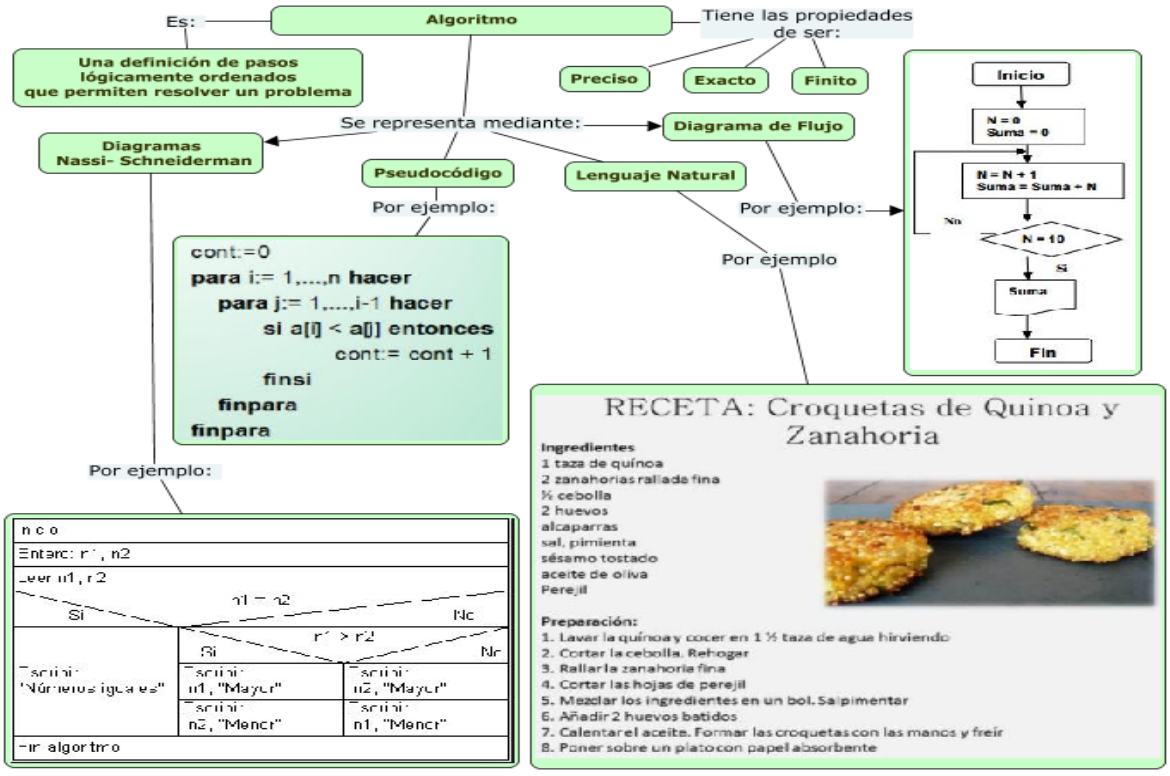
9. Muchos bancos y cajas de ahorro calculan el interés de las cantidades depositadas por los clientes diariamente en base a las siguientes premisas. Un capital de 1000 pesos, con una tasa de interés del 6 por ciento da un interés en un día de 0.06 pesos multiplicado por 1000 y dividido por 365. Esta operación producirá 0.16 pesos de interés y el capital acumulado será 1000.16. El interés para el segundo día se calculará multiplicando 0.06 por 1000.16 dividido por 365. Analizar para lograr una solución que reciba tres entradas: el capital a depositar, la tasa de interés y la duración del depósito en semanas (7 días) y que calcule el capital total acumulado al final del periodo especificado

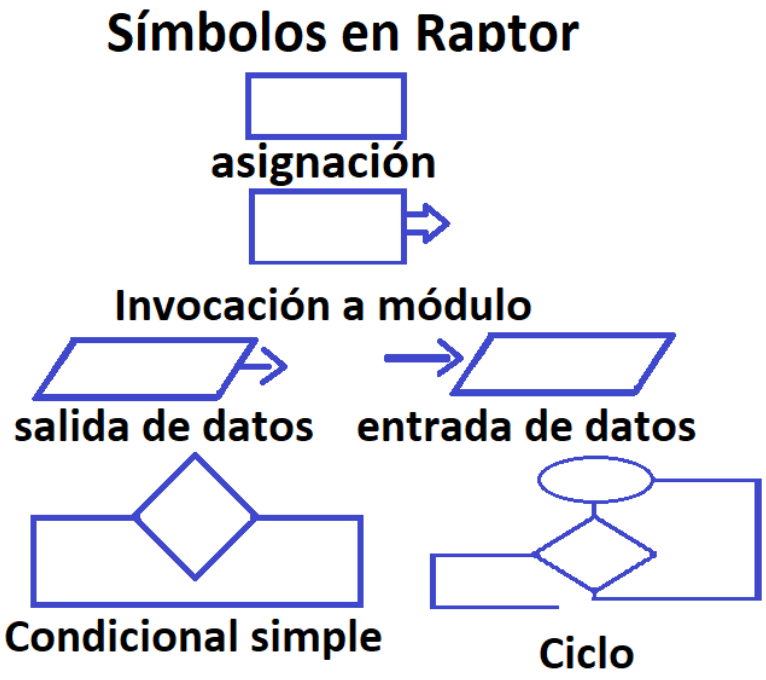
Análisis	Pasos a seguir para resolver (Puede hacer un algoritmo)
Datos de entrada: **Proceso:** **Datos de salida:**	

Capítulo 2
Los Algoritmos, su representación y codificación

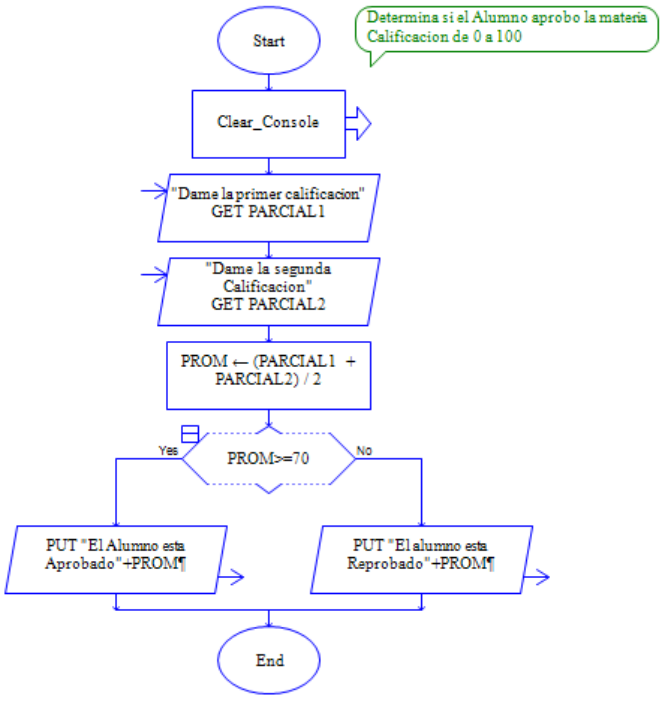
OBJETIVO: El alumno usará las reglas de sintaxis básicas del lenguaje C para lograr la entrada y salida de datos y el uso de todas las estructuras de control .

2.1 DESCRIPCIÓN DEL TEMA





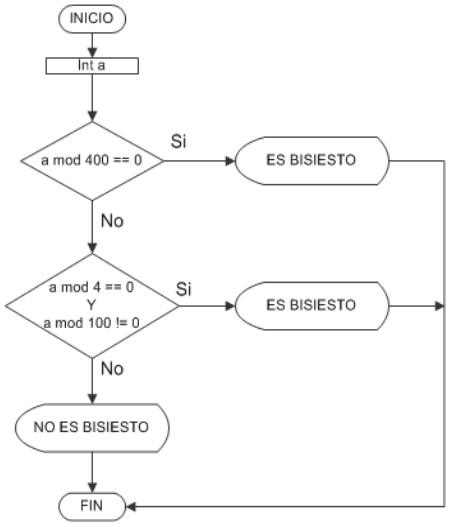
2.1.1 Cuestionarios y ejercicios

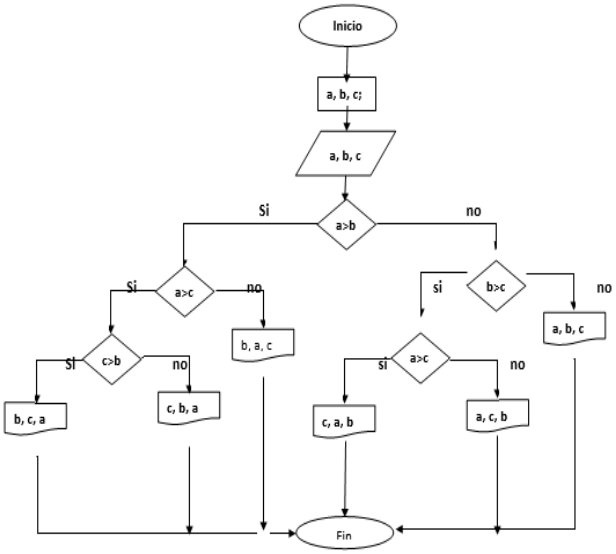



```
//Determina si el alumno aprobó una
//materia con calificaciones de 0 100
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int parcial1, parcial2;
float prom;
main(){
system("cls");
printf("Dame la primer calificación \n");
scanf("%d",&parcial1);
printf("Dame la segunda calificación
\n");
scanf("%d",&parcial2);
prom=(parcial1+parcial2)/2;
if (prom>=70){
printf("El alumno esta aprobado \n");}
else { printf("El alumno NO esta aprobado
\n");}
system("pause");
}
```

1. A partir de los siguientes algoritmos completa los datos faltantes y el código necesario

Análisis	Algoritmo	Codificación

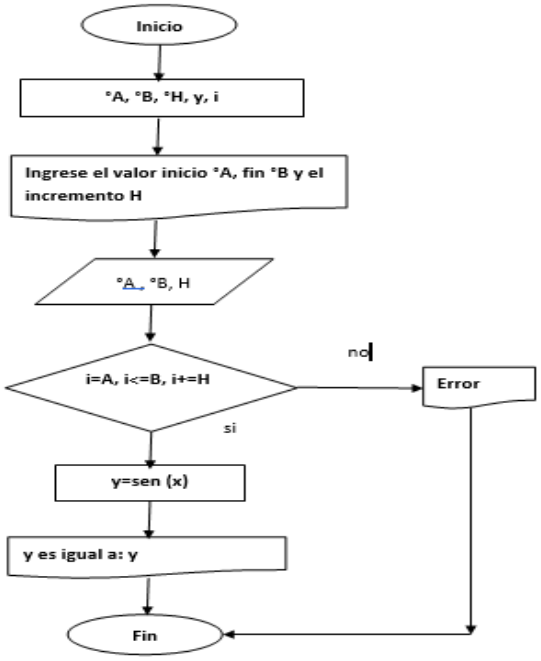
Datos de entrada:
 	
Proceso:
 	
Datos de salida:
		`//Ejemplo de líneas de código`
		`#include <stdio.h>`
		`#include <stdlib.h>`

Datos de entrada: **Proceso:** **Datos de salida:**		

Datos de entrada:

Proceso:

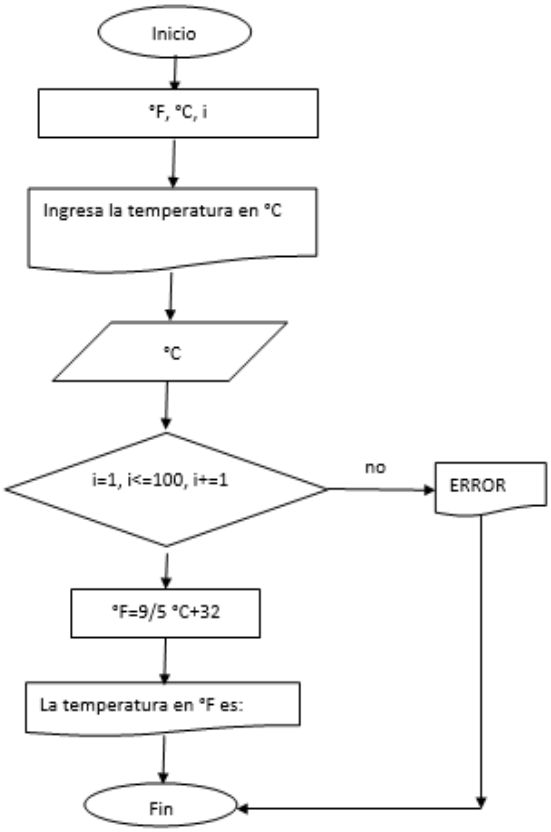
Datos de salida:



Datos de entrada:

Proceso:

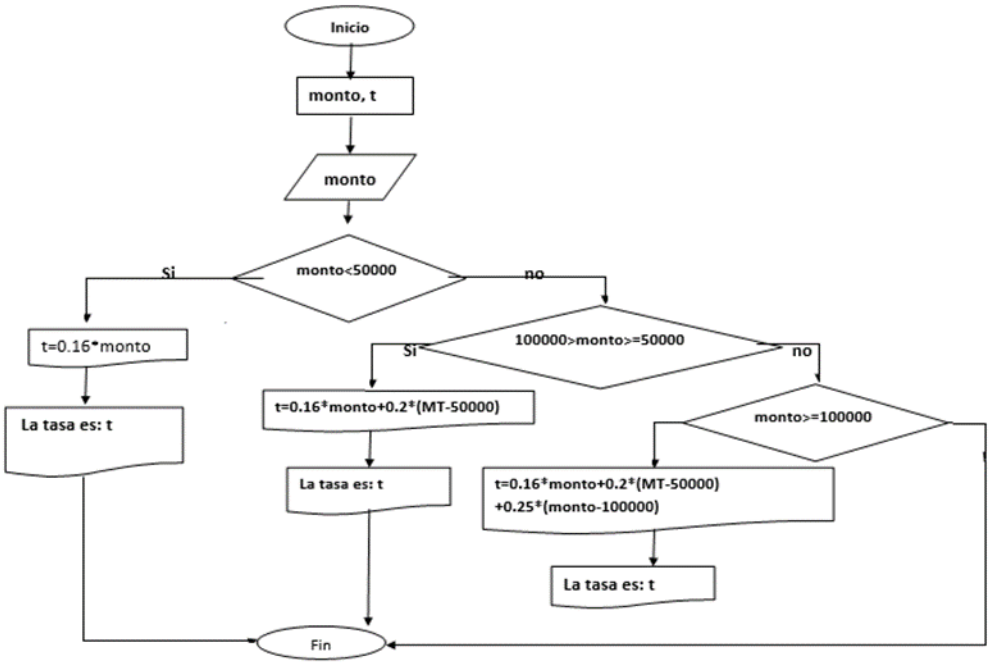
Datos de salida:



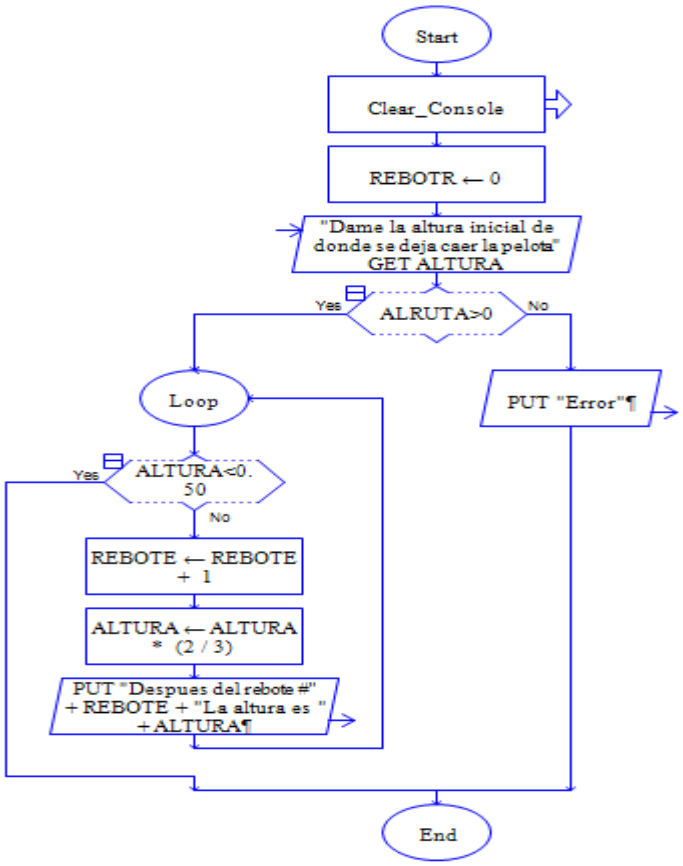
Datos de entrada:

Proceso:

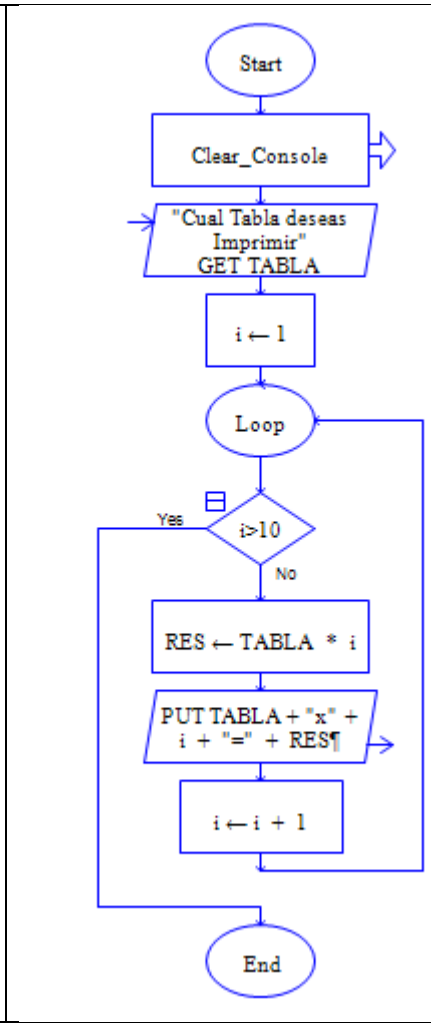
Datos de salida:



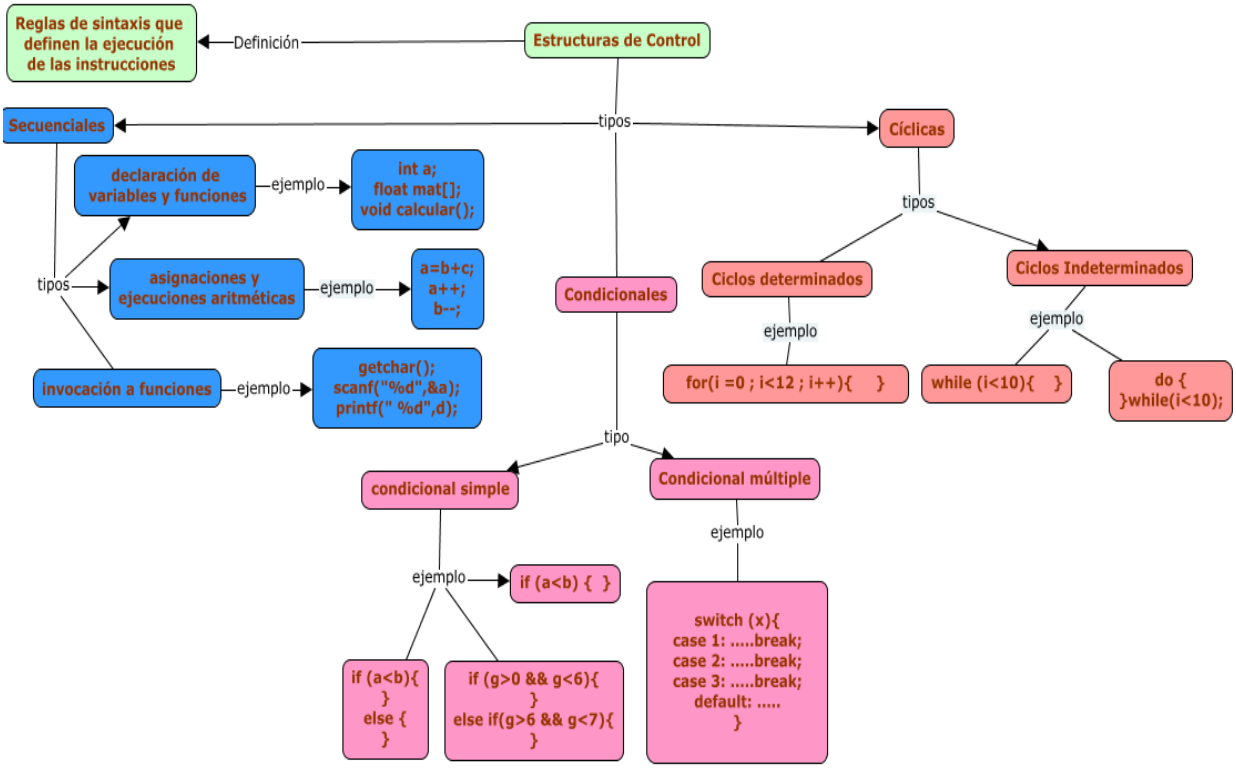
2. Analiza los pseudocódigos y representarlos como diagramas de flujo, puedes hacerlo con dibujo o usando software de edición de algoritmos como DFD o raptor

Pseudocódigo	Diagrama de Flujo en Raptor o DFD
Inicio Leer el valor de E Si E es menor que 0 mostrar error Si E no es menor que 0 Entonces a 220 restarle el valor de E y dividirlo entre 10 y almacenarlo en P Mostrar P Fin	
	

Inicio Leer el valor de x Si x es menos o igual a 0 mensaje error Si x es mayo que 0 Entonces Al valor de x multiplicarlo por 15 y dividirlo entre 100 y almacenarlo en d Al valor de x restarle el de d y almacenarlo en r Mostrar r Fin	
Inicio Leer los valores de a, b, c Si los valores de a, b, c son menores que 0 y mayores que 10 mostrar error Si los valores de a, b, c están entre 0 y 10 Entonces Sumar los valores de a, b, c y almacenarlos en s Al valor de s dividirlo entre 3 y almacenarlo en p Al valor de p multiplicarlo por 0.55 y almacenarlo en r Leer los valores de e, t Si los valores de e, t son menores de 0 y mayores de 10 mostrar error Si los valores de e, t están entre 0 y 10 Al valor de e multiplicarlo por 0.3 y almacenarlo en u Al valor de t multiplicarlo por 0.15 y almacenarlo en z Sumar los valores de r, u, z y almacenarlo en k Mostrar k Fin	


Start
Clear_Console
"Cual Tabla deseas Imprimir"
GET TABLA
i ← 1
Loop
Yes
i>10
No
RES ← TABLA * i
PUT TABLA + "x" + i + "=" + RES¶
i ← i + 1
End

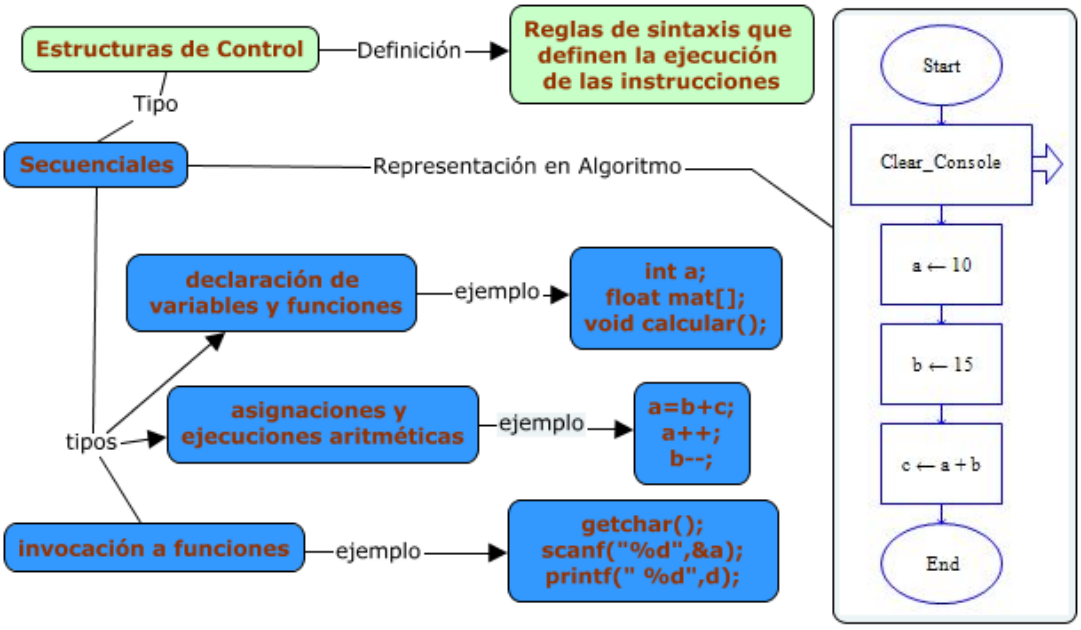
2.2 Las Estructuras de Control



Las estructuras de control son reglas de sintaxis del lenguaje de programación que dan orden a las sentencias de ejecución, las reglas de sintaxis se clasifican en tres grupos:

- Las sentencias secuenciales, son aquellas sentencias que se ejecutan una tras otra y que generalmente se agrupan en bloques de código llamados funciones. De forma general la declaración de funciones y variables, la invocación de funciones y las asignaciones son de tipo secuencial.
- Las sentencias condicionales: las estructuras condicionales permiten cambiar la ejecución del programa con base en el resultado de alguna expresión lógica, se tienen la condicional simple, la cual permite conmutar el control en dos posibles rutas y la condicional múltiple que permite conmutar la ruta en más de dos caminos de ejecución
- Las sentencias cíclicas: son estructuras que permiten la repetición de un bloque de sentencias n veces, se tienen los ciclos determinados (se conoce el número de iteraciones) y los ciclos indeterminados (el programador no sabe con anterioridad el número de iteraciones que se ejecutaran).

2.2.1 ESTRUCTURAS SECUENCIALES



Para definir identificadores en C debemos aplicar las siguientes reglas:
1 Se debe definir el tipo de dato seguido del nombre del identificador, por ejemplo:
int a, x,tel;
char tele[15];

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

Los tipos de datos se caracterizan por tener una magnitud propia, la cual es determinada por el compilador, de forma general las dimensiones en bytes son las siguientes:

Tipo de dato soportados por el lenguaje	Palabra reservada	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	*	

Las funciones printf y scanf requieren un indicador del tipo de dato que debe procesar, para lo cual se han definido dos letras que cumplen con esta función el símbolo % y una letra identificadora para cada tipo de dato. Así por ejemplo si se usara la función printf para mostrar un valor entero se usará de la siguiente forma:

printf(" el valor de x es : %d", x);

si se lee un entero con scanf

Código de formato

%d	Entero decimal
%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;

Tipo de dato compuestos	**Palabra o símbolo**
arreglo	**[ ]**
estructura	**struct**
unión	**union**
archivo	**FILE**

Sistemas Numéricos en C

octal	0 el número en octal
hexadecimal	0x el número en hexadecimal
decimal	el número en decimal

Operadores Aritméticos

Símbolo	Operadores binarios	uso
+	suma	a+B
-	resta	a-B
*	multiplicación	a*B
/	división	a/B
%	módulo	a%B

Asignación compuesta	Equivalencia
a+ = x	a=a + x
a- =x	a=a - x
a* =x	a=a * x
a/ =x	a=a / x
a%=x	a=a % x

Operadores Unarios

a++	incremento
a--	decremento
a>>3	corrimiento a la derecha
a<<3	corrimiento a la izquierda

Operadores relacionales

A < B	menor que
A > B	mayor que
A <= B	menor o igual que
A >= B	mayor que
A == B	Igual que
A != B	diferente que

Operadores Lógicos

!	Negación
||	Operador o
&&	Operador y

Operadores	Asociatividad	Tipo
( )	de izquierda a derecha	paréntesis
(tipo) ++ -- + -	de derecha a izquierda	unario
*** / %**	de izquierda a derecha	multiplicativo
. + -	de izquierda a derecha	aditivo
< <= > >=	de izquierda a derecha	relacional
. == !=	de izquierda a derecha	igualdad
? :	de derecha a izquierda	condicional
. = += -= *= /= %=	de derecha a izquierda	de asignación

Ejemplo de aplicación: Necesitamos un programa que calcule el área de un circulo

Análisis: Se sabe que para calcular el área de un circulo se requiere conocer el radio y se usa la constante PI, por lo tanto, el radio es un dato de entrada y se conoce el valor de PI= 3.1416.
El cálculo consiste en multiplicar la constante PI por el radio al cuadrado.

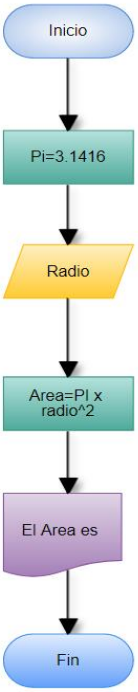
Conociendo esta información ya se puede identificar lo siguiente:

- Se requieren dos variables radio, área de tipo float para recibir datos numéricos
- Se requiere asignar el valor de PI a una constante.
- Con las variables preparadas se necesitará solicitar el radio y asignarlo a la variable radio
- Asignar a área el cálculo: área=PI *radio *radio
- Mostrar en pantalla el valor de area

En la siguiente tabla se documentan y representan los pasos que realizamos en forma de algoritmo y se codifica con C

Programa: Diseñe un programa que calcule el área de un circulo.

Análisis	Diseño (Diagrama de flujo)	Desarrollo (Codificación en C)

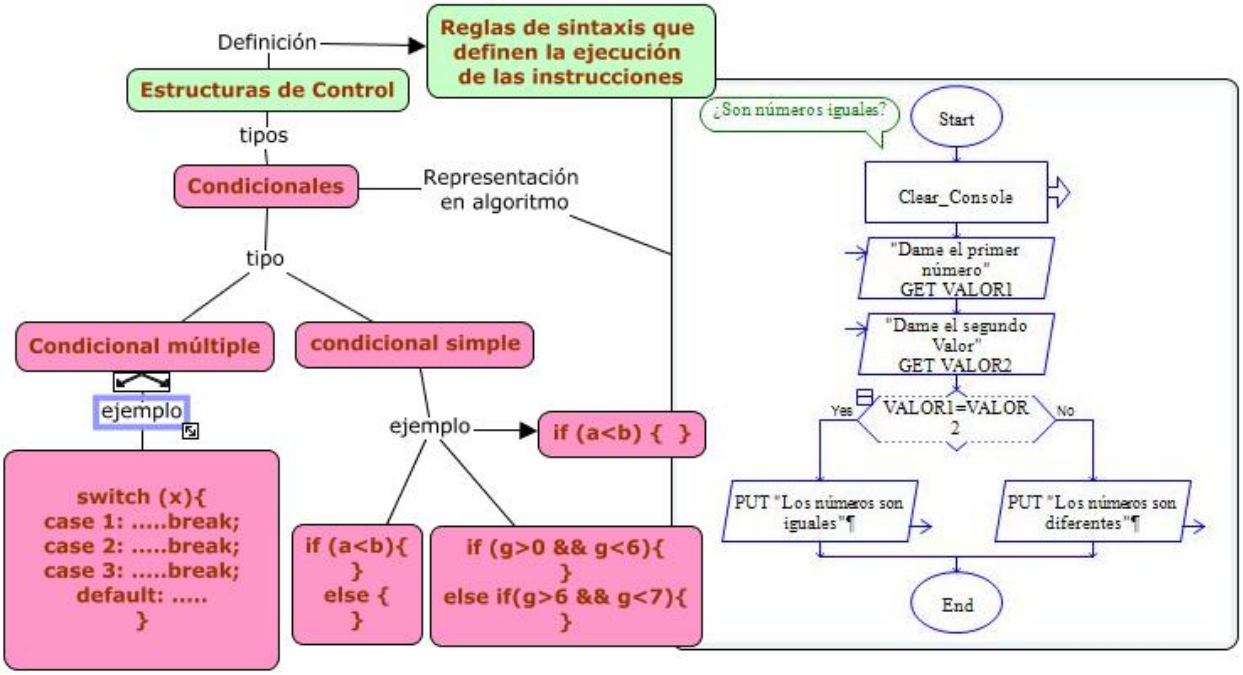
Entrada
float Pi= 3.1416
float radio
 	
Proceso
area =pi* radio * radio
 	
Salida:
imprimir **a**rea
		`include <stdio.h>`
		`#include <stdlib.h>`
		`#include <math.h>`
		`#define pi 3.1416`
		`int main(int argc, char** argv) {`
		`  float area,radio;`
		
		`  printf("**Area de un Circulo**\n");`
		`  printf("Ingresa el area:\n");`
		`  scanf("%f",&radio);`
		`  area=pi*pow(radio,2);`
		`  scanf("El area del circulo es:\n");`
		`  printf("%f",area);`
		`  return (EXIT_SUCCESS);`
		`}`

Con base en el ejemplo anterior dibuje el algoritmo y subraye con rojo la declaración de variables, con azul las asignaciones, con verde las llamadas a función y con rosa la inclusión de librerías

Algoritmo		
	Sentencias en C	Explicación de Código
	`#include <stdlib.h>`	`archivo con funciones estándar`
	`#include <stdio.h>`	`archivo con funciones de entrada y salida de datos`
	`#include <string.h>`	`archivo con funciones de manejo de cadenas`
	`main(){`	`programa principal`
	`int t1,t2,t3,tr,comp;`	`variables tipo entero`
	`char var1[40], var2[40], var3[40],texto[40];`	`variables tipo cadenas de caracteres`
	`system("color F1");`	`uso de color de fondo y de texto en la pantalla`
	`strcpy(var1,"Texcoco");`	`copia del texto "Texcoco" en la variable v1`
	`strcpy(var2,"Centro Universitario UAEM ");`	`copia del texto entre comillas en la variable v2`

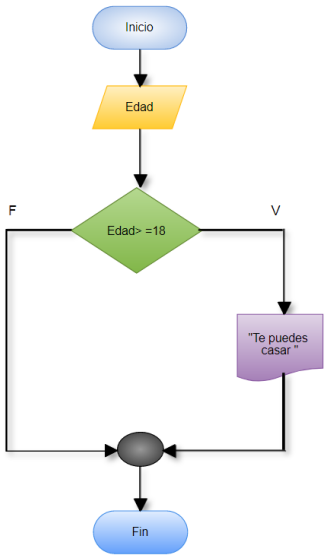
	strcpy(var3,"Ingeniería en Computación");	copia del texto entre comillas en la variable v3
	printf("Introduce una cadena de texto \n");	Indicación al usuario
	scanf("%[^\n]",&texto);	lectura de un texto de una línea
	t1=strlen(var1);	asignación del tamaño de la variable var1 a la variable t1
	t2=strlen(var2);	asignación del tamaño de la variable var2 a la variable t2
	t3=strlen(var3);	asignación del tamaño de la variable var3 a la variable t3
	system("cls");	limpieza de pantalla
	printf("Tamaño var1: %d \n Tamaño var2: %d \n Tamaño var3: %d \n",t1,t2,t3);	se muestran los tamaños de las cadenas
	strcat(var1,var2);	concatena la variable var2 a la variable var1
	printf("Texto resultado var1: %s \n",var1);	se muestra el resultado de la concatenación
	printf("Texto de :%s \n",texto);	se muestra el texto
	t1=strlen(var1);	se actualiza el tamaño de la variable var1 en la variable t1
	printf("Tamaño var1: %d \n",t1);	se muestra el valor actualizado
	comp=strcmp(var1, "texcocoCentro Universitario UAEM ");	se compara la variable var1 con el texto entre comillas, el resultado se almacena en comp
	printf("Comparacion: %d \n", comp);	se muestra el resultado de la comparación
	system("pause");	se genera una pausa
	}	

2.2.2 Estructuras condicionales

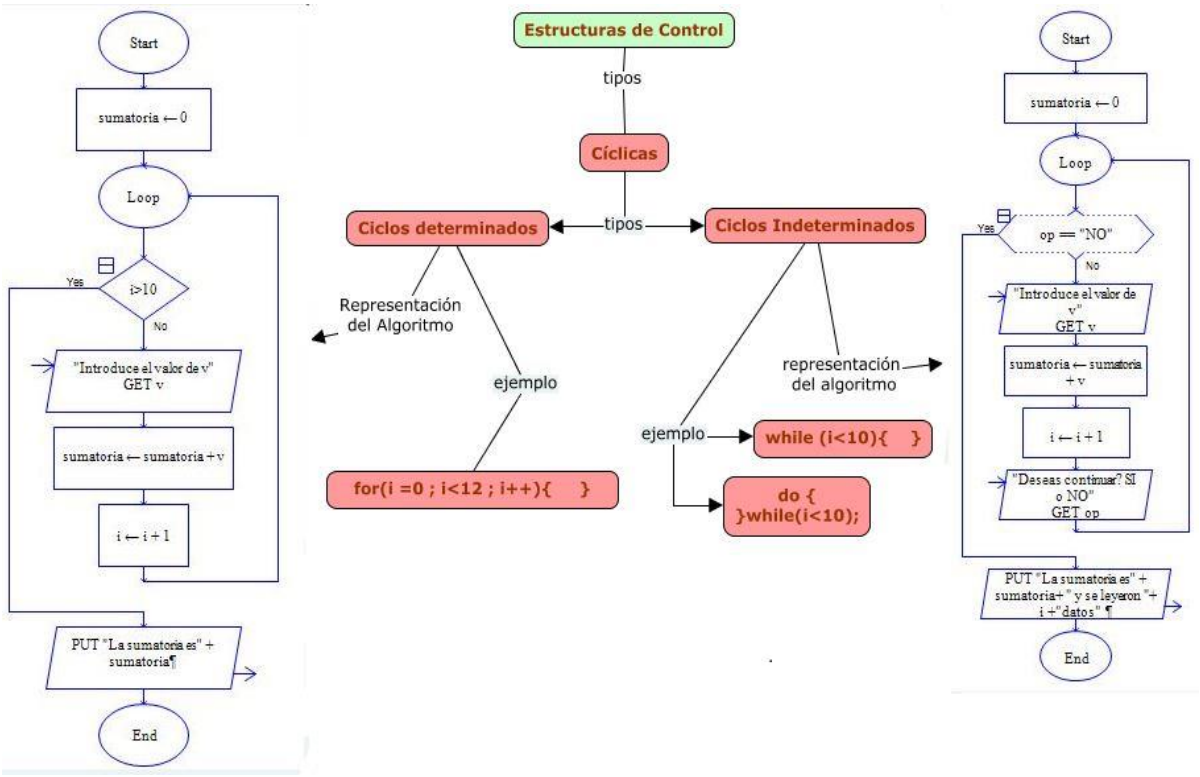


Ejemplo de **estructura de control condicional** simple (if):

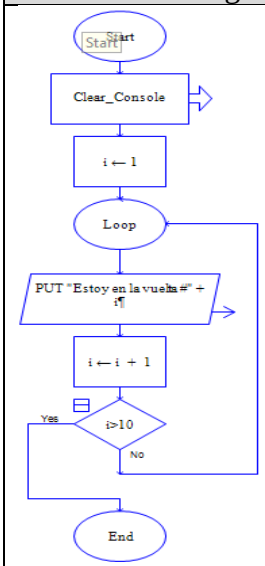
Programa: Una persona se puede casar siempre y cuando su edad sea mayor o igual a 18 años, imprimir "Te puedes casar".		
Análisis	Diseño (Diagrama de flujo)	Desarrollo (Codificación en C)

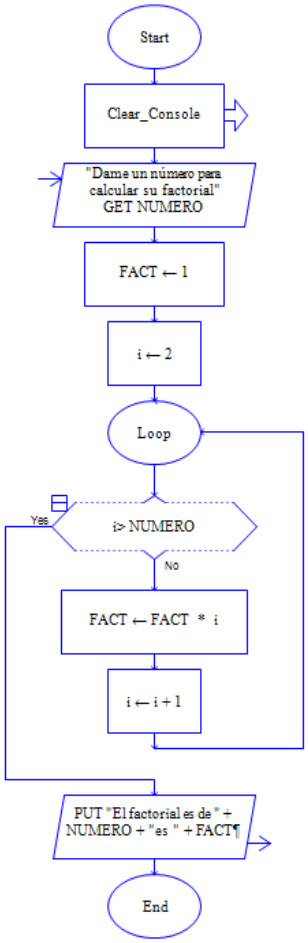
Entrada
int Edad
Proceso
Si Edad>=18
"Te puedes casar"
 	
Salida
imprimir "Te puedes casar "
		`#include <stdio.h>`
		`#include <stdlib.h>`
		`int main(int argc, char** argv) {`
		`  int Edad;`
		
		`  printf("Ingresa tu edad:");`
		`  scanf("%d",&Edad);`
		`  if(Edad>=18)`
		`  printf("Te puedes casar");`
		`  return (EXIT_SUCCESS);`
		`}`

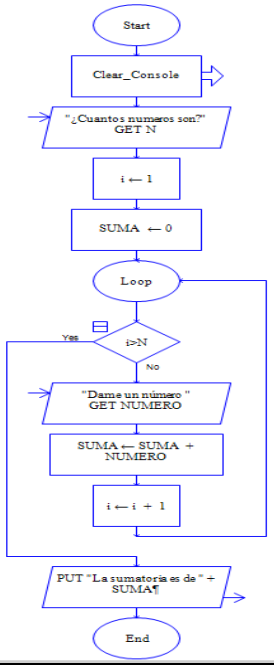
2.2.3 Estructuras cíclicas

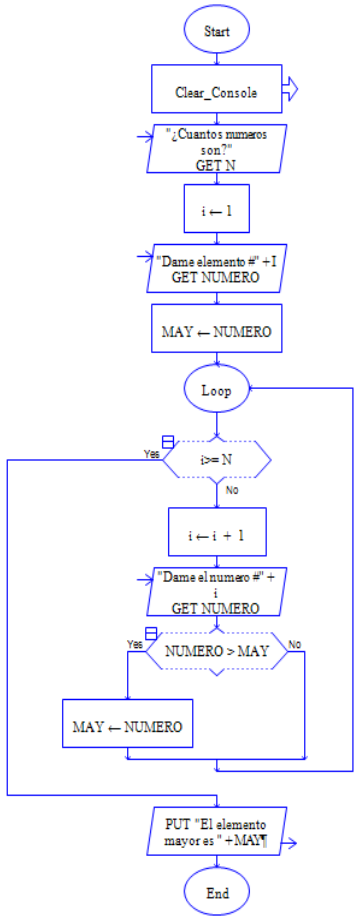


Los ciclos pueden codificarse con alguna estructura cíclica equivalente, lo importante es que se use una variable de control

Algoritmo	Código con for	Código con While
		



Algoritmo	Código con for	Código con While
		



2.3 Cuestionario y ejercicios

1. ¿Para qué se usa la directiva # include?
2. ¿Se pueden anidar los comentarios?
3. ¿Los comentarios pueden ser más grandes que una línea?
4. ¿Qué es un archivo de inclusión?
5. ¿Qué reglas hay que seguir para la creación de nombres para las variables y constantes?
6. ¿Cómo se llama a un grupo de enunciados del C encerrados entre llaves?
7. ¿Cuál es el único componente obligatorio de todo programa en C?
8. ¿Cómo se añaden comentarios al programa y para que se usan?
9.

Ejercicios

10. Haga un programa que genere 30 números aleatorios entre 1-60 y que muestre sólo los múltiplos de 4

11. Con los valores definidos para A = 13, B= 45, C= 21 evalué las operaciones y especifiqué los resultados de cada variable y en caso se ser operaciones lógicas indiqué el resultado con un tache en la columna falso o verdad

Operaciones	A	B	C	Cálculo	Verdad	Falso
C== A/B* (2*B/C)	13	45	21	1.2380		X
A*=45	585	45	21	585		
C - =B						
A= (C/4) %B						
B%=5						
C*=B*C*A+(C-A)						
C== A/((B*2*B)/C)						
A*= 45+C(-A)						
C-=A						
A= (C/5.5)*B						
B<=5						
C>=B*C*A +(C-A)						
C != A*B(2+B/C)						
A*=45						
C-=B						
B != (C-B)						
B>=5-A						
C<=B*C*A+(100-A)						
C= A/C*(2*B/C)						

…

Capítulo 3
Arreglos

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

3.1 ARREGLOS UNIDIMENSIONALES

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};

```
#include <time.h> //tiene la función time que toma la hora de la PC
#include <stdlib.h> //tiene la definición de la función rand (random) y srand
(generador de semilla para aleatorios)
#include <stdio.h> //archivo que contiene funciones de entrada y salida de
datos
main (){ //función principal
    int numV, alea; //variables enteras
    srand(time(NULL)); // la función time toma el tiempo de la PC, y lo envía
como la semilla del generador de aleatorios que es srand
    int cantidad, semilla; // variables enteras
  printf( "Cuantos números quiere generar?" ); // solicitud de datos al
usuario
  scanf("%d",&numV); getchar();// leer de teclado el valor del usuario
  // ciclo que inicia en 1 y funciona mientras el contador sea menor o igual
al valor leído del teclado
  for (int contador =1; contador <= numV; contador++){
      printf( "%d \n",(1 + rand() % 20)); //mostramos en pantalla el número
generado en un rango del 1 al 20 con la función rand + salto de línea para
separarlos
      getchar();  } // fin de ciclo for
getchar();    } //fin de main.
```

3.2 Cuestionarios y ejercicios

1. Haga un programa que guarde en un arreglo bidimensional de 5 x 5 valores aleatorios entre 1- 10 y que muestre la sumatoria de los valores almacenados

Haga un programa que permita guardar por el usuario los valores de 2 matrices de 3 X 3 y obtenga la suma de las matrices, la multiplicación de las dos matrices por un escalar (número entero introducido por el usuario).

2. Por medio de matrices haga un programa que permita almacenar 15 nombres, que almacene 15 calificaciones para 5 materias y 15 promedios de esas 5 materias. El programa permitirá que el usuario consulte el nombre las calificaciones y el promedio de alguno de los 15 espacios guardados.
3. Un arreglo es declarado con el enunciado int arreglo [2] [3] [5] [8];
   ¿Cuántos elementos tiene el arreglo?
4. 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 arreglo almacenaremos las calificaciones de 5 materias del cada uno de los alumnos
   c) En el tercero almacenaremos el promedio de cada alumno.

El programa debe permitir insertar información, consultar los datos almacenados y modificar con base en el índice del arreglo. Ejemplo del contenido y estructura de los arreglos necesarios

Nombre de Alumno
Luis Méndez Huerta

Calificaciones				
8.8	9.2	6.7	7.9	6.9

Promedio de calificaciones
7.9

5. 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 almacenaran las ventas, su comisión y su sueldo.

El programa deberá incluir la introducción de datos y la muestra de los datos en forma de tabla:

Nombre de E
Ana Pérez

Departamento
Ventas

datos		
8500	2.125	5.125

6. Haga un programa que use tres arreglos de tipo flotante, en ellos se debe almacenar:
   a. en el primero el radio de 50 círculos
   b. en el segundo se debe guardar el área de los 50 círculos
   c. en el tercero se almacenará el volumen de 50 cilindros considerando que las alturas serán generadas de forma aleatoria en un rango de 1 a 20 cm.
   d. El programa debe mostrar los datos de cada arreglo y debe permitir modificar

Radio de Circulo
6.5

Área de Círculo
45.8

Volumen de cilindro
448.78

7. Explique línea a línea el significado de las siguientes instrucciones:

```
1)  #include <stdio.h>
2)  #include <string.h>
3)  #include <windows.h>
4)  #define tam 40
5)  char n1[tam], n2[tam], n3[tam], naux[80];
6)  int tam1,tam2;
7)  main(){
8)  printf("Dime el primer nombre \n");
9)  scanf("%[^\n]s",&n1); getchar();
10) printf("Dime el segundo nombre \n");
11) scanf("%[^\n]s",&n2);getchar();
12) tam1=strlen(n1);
13) tam2=strlen(n2);
14) if (tam1<=tam2)printf ("el nombre menor es %s con %d letras \n \n",n1,tam1);
15) else  printf ("el nombre menor es %s con %d letras \n \n",n2,tam2);
16) strcpy(n3,n1);
17) strcat(n1,n2);
18) printf ("el nombre unido es %s con %d letras \n",n1,strlen(n1));
19) printf ("\n \n");
20) strcpy(naux,n2);
21) strcpy(n2,n1);
22) strcpy(n1,naux);
23) printf ("Los nombres son\n");
24) printf(" Nombre \t|  Tamaño\t| \n\n");
25) printf("%s \t |%d \t | \n\n",n1,strlen(n1));
26) printf("%s \t |%d \t | \n \n",n2,strlen(n2));
27) printf("%s \t|%d \t | \n \n",n3,strlen(n3));
28) getchar(); getchar();
29) system("PAUSE");
30) }
```

Capítulo 4
Programación Modular

4.1 Funciones

C permite al programador definir sus propias funciones que realicen determinadas tareas. El uso de funciones definidas por el programador permite dividir un programa grande en un cierto número de componentes más pequeñas, cada una de las cuales con un propósito único e identificable.

Todo programa en C consta de 1 o más funciones. Una de estas funciones se debe llamar main. Si un programa contiene varias funciones, éstas deben ser independientes unas de otras.

Generalmente, una función procesará la información que le es pasada desde el punto del programa en donde se accede a ella y devolverá un solo valor, es decir, cuando se accede a una función desde alguna parte del programa, se ejecutan las instrucciones de que consta.

La información se le pasa a la función mediante unos identificadores llamados argumentos (también denominados parámetros) y es devuelta por medio de la instrucción return. Sin embargo, algunas funciones aceptan información, pero no devuelven nada (la función de biblioteca printf), mientras que otras funciones trabajan sobre varios valores (la función de biblioteca scanf).

DEFINICIÓN DE FUNCIONES.

La definición de una función tiene dos componentes: *la primera línea* (incluyendo las declaraciones de los argumentos) y el *cuerpo de la función.*

1. La primera línea contiene la especificación del tipo de valor devuelto por la función, seguido del nombre de la función. En términos generales, la primera línea se puede escribir así:
   Tipo-de-dato nombre (tipo 1 arg 1, tipo 2 arg 2,… tipo n arg n)

Donde:

- [1]*Tipo-de-dato: representa el tipo de datos del valor que devuelve la función.*
- [2]*Nombre: el nombre de la función.*
- [3]*tipo 1, tipo 2,…, tipo n representan los tipos de datos de los argumentos o parámetros (representan los nombres de los elementos que se transfieren a la función).*

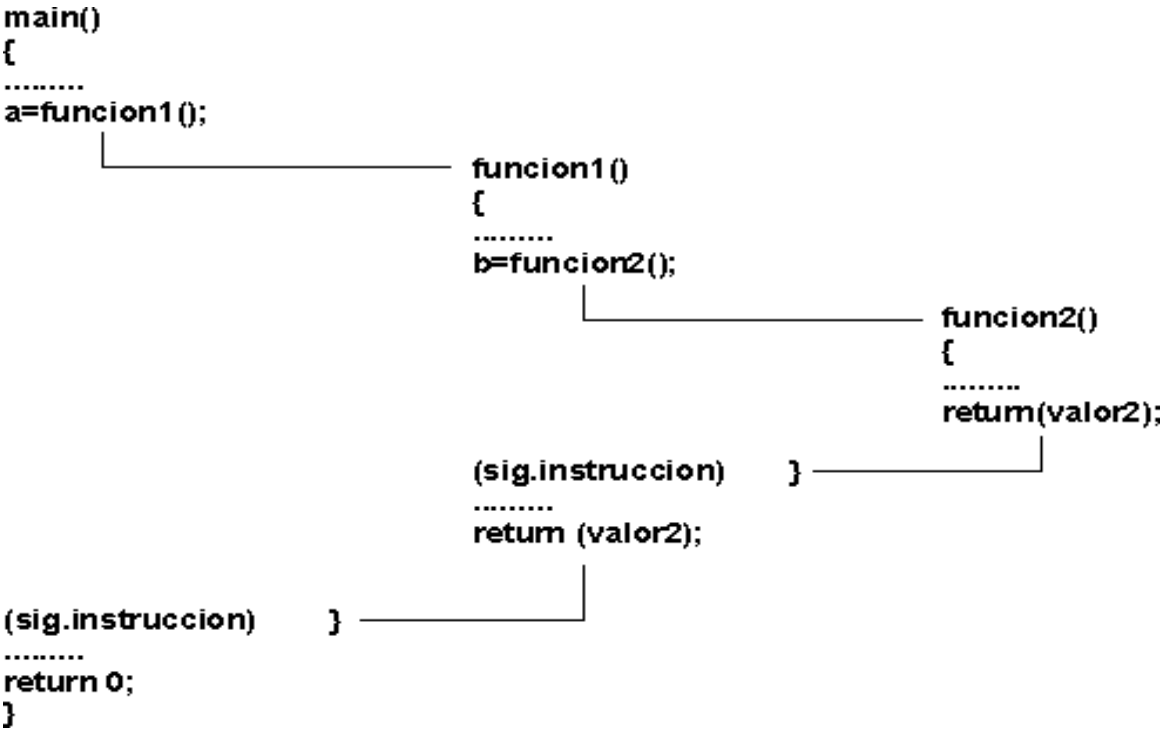
2. El cuerpo de la función es una instrucción compuesta que define las acciones que debe realizar ésta.
3. Se devuelve información desde la función hasta el punto del programa desde donde se llamó mediante la instrucción return. La instrucción también hace que se devuelva el control al punto de llamada.

   [4]*return expresión;*

Ejemplo:

```
#include <stdio. h>
char[1] minusc_a_mayusc[2] (char c1[3])          /*definición de la función*/
 {
   char c2;
    if (c1>='a' && c1<='z') c2='A' +c1-'a';
   else  c2 = c1;  /*cuerpo de la función*/
    [4]return (c2);
  }
main () {       /*inicio de la función main*/
char minusc, mayusc;
printf("Por favor, introduce una letra minúscula : ") ;
scanf ("%c", &minusc);
mayusc = minusc_a_mayusc (minusc[5]);                /*llama a la función*/
printf ( "\n La mayúscula equivalente es %c \n \n ", mayusc);
}
```

Vemos en este ejemplo, la posibilidad de múltiples llamados a funciones, una llama a otra, que a su vez llama a otra (conocido como recursividad), la cual llama a otra, etc, dando un esquema de flujo de programa de la forma:



ACCESO A UNA FUNCIÓN

Se puede acceder a una función especificando su nombre, seguido de una lista de argumentos encerrados entre paréntesis y separados por comas[5]. Si la llamada a la función no requiere ningún argumento, se debe escribir a continuación del nombre de la función un par de paréntesis vacíos.

Si la función devuelve un valor, el acceso a la función se suele escribir como una instrucción de asignación; esto es:

Y = polinomio (x);

Este acceso a la función hace que a la variable se le asigne el valor devuelto por la función

EJEMPLO

Linea de Código	Comentario explicativo
#include <stdio.h>	//inclusión de archivo con funciones de entrada y salida de datos
#include <stdlib.h>	//inclusión de archivo con funciones estándar
#include <windows.h>	//inclusión de archivo con funciones que se comunican con el sistema operativo
#include <string.h>	//inclusión de archivo con funciones de manejo de cadenas
struct Alumno {	//Declaración del registro Alumno formado de tres campos
int matricula;	// matricula
char nombre[20];	// nombre

int nota;	// nota
} ;	// fin de registro
Alumno ingresoDatosXConsola();	// prototipo de la función de ingreso de datos
void muestraDatosAlumno(Alumno a, int v);	// prototipo de la función muestra datos
float promedioNotas(Alumno a[], int v);	// prototipo de la función calcula promedio de notas
int menu();	// prototipo de la función que muestra un menú de opciones
Alumno ingresoDatosXConsola(){	//Lectura de datos por consola
Alumno a;	//variable tipo Alumno
int matricula, nota;//	//variables temporales para introducción de datos
char nombre[20];	// variable para inserción de datos
printf("Ingrese nuevo matricula: "); scanf("%d", &matricula); getchar();	//solicitud en pantalla y lectura del teclado del campo matricula
printf("Ingrese nuevo nombre: "); scanf("%[^\n]s", &nombre);	// solicitud en pantalla y lectura del campo nombre
printf("Ingrese nueva nota: "); scanf("%d", &nota);	// solicitud en pantalla y lectura de nota
a.matricula = matricula;	//asignación al campo matricula
strcpy(a.nombre, nombre);	// copia a campo nombre
a.nota = nota;	// asignación a nota
return a;	// retorno de la variable a con los datos en sus campos
}	// fin de función
void muestraDatosAlumno(Alumno a[], int v){	// se recibe un arreglo de registros tipo Alumno y un entero
int i;	// variable contador para control de ciclo
for (i=0;i<v;i++){	//ciclo para recorrer a los registros con valores
printf("**** \n");	//en pantalla se muestra un salto de línea
printf("**** Matrícula del alumno: %d ****\n", a[i].matricula);	// se muestra en pantalla la matricula
printf("**** Nombre del alumno: %s ****\n",a[i].nombre);	// se muestra en pantalla el nombre
printf("**** Nota: %d ****\n",a[i].nota); // se muestra la nota	printf(" \n \n"); } // se muestran dos saltos de línea
system("pause"); // se genera una pausa	
}	// fin de función
int menu(){	// inicio de función menú
int x;	//variable entera
do{	// inicio de ciclo "hacer"
printf("**** \n Este programa permite ingresar los datos de diez alumnos \n \n");	// información al usuario mediante pantalla
printf("**** Selecciona tu preferencia \n");	// indicación al usuario
printf("**** 1: Ingresar alumnos \n");	// opción de ingresar datos

printf("**** 2: Mostrar datos de los alumnos \n");	// opción de mostrar datos
printf("**** 3: Calcular promedios de alumnos\n");	// opción de calcular promedios
printf("**** 4: Salir\n \n");	// opción de salir
scanf("%d",&x);	// lectura de selección del usuario
if(x<1 || x>3) printf(" \n *** opción no valida inserte un nuevo dato **** \n \n ");	// validación de entrada
} while (x<1 || x>4);	// control de ciclo, funciona mientras no se reciba un dato válido
return x;	// se entrega a la función principal la opción elegida
}	// Fin de la función
float promedioNotas(Alumno a[], int v){	// se recibe un arreglo de registros tipo Alumno y un entero
int i,sum=0; float prom;	// variables locales
for (i=0;i<v;i++) sum+=a[i].nota;	// ciclo que recorre el arreglo de registros y acumula el valor de las notas
prom=sum/v;	//cálculo del promedio
return prom;	// entrega a la función principal el promedio calculado
}	// fin de función
main(){	// función principal
int op=0,i=0;	// declaración de variables tipo entero
do {	// ciclo
Alumno reg[10];	// arreglo para 10 registros
op= menu();	// invocación de la función menú para conocer la opción del usuario
switch(op){	// conmutación de línea de ejecución
case 1: reg[i] = ingresoDatosXConsola(); i++; break;	//invocación de ingreso de datos, asignación al arreglo y conteo de registros
case 2: muestraDatosAlumno(reg,i); break;	// invocación a la función que muestra los datos guardados
case 3: printf (" \n \n El promedio de las notas es %f: ",promedioNotas(reg,i));break;	// cálculo y muestra del promedio
}	// fin de condicional múltiple
}while (i<10 || op!=4);	// control de ciclo funciona mientras i es menor a 10 o la opción es diferente de salir
}	// fin de función principal

4.2 Cuestionarios y ejercicios

1. Con base en el siguiente ejemplo resuelve las secciones faltantes:

Declaración de Función	Codificación	Invocación de la función

1. Calcula el perímetro de un triángulo escaleno no recibe datos, devuelve el perímetro.
 	
Prototipo: float calPeri();
	`void calPeri(){`
	`float l1,l2,l3;`
	`printf("Introduce el lado 1 \n");`
	`scanf("%f",&l1);`
	`printf("Introduce el lado 2 \n");`
	`scanf("%f",&l2);`
	`printf("Introduce el lado 3 \n");`
	`scanf("%f",&l3);`
	`return (l1+l2+l3);`
	`}`
		`main(){`
		
		`printf("En este programa se calcula el perímetro de un triángulo \n");`
		`printf("%f",calPeri());`
		
		`}`

2. Calcula el perímetro de un triángulo escaleno recibe tres datos, devuelve el perímetro.
 	
Prototipo:
float calcPeri(float a, float b, float c);
	`void calcPeri(float a, float b, float c){`
	`return (a+b+c);`
	`}`
		`#include <stdio.h>`
		`float calcPeri(float a, float b, float c);`
		`main(){`
		`float l1,l2,l3, p;`
		`printf("En este programa se calcula el perímetro de un triángulo escaleno \n");`
		`printf("Introduce el lado 1 \n");`
		`scanf("%f",&l1);`
		`printf("Introduce el lado 2 \n");`
		`scanf("%f",&l2);`
		`printf("Introduce el lado 3 \n");`
		`scanf("%f",&l3);`
		`p= calcPeri(l1,l2,l3);`
		`printf("%f",p);`
		
		`}`

3. Calcula el área de un polígono regular, recibe tres datos (el número de lados, la dimensión del lado, la apotema), devuelve el área. Prototipo:		

4. Recibe un número, devuelve el valor recibido multiplicado por un segundo valor que es solicitado dentro de la función. Prototipo:		
5. Recibe una cadena de caracteres y la muestra al revés. Prototipo:		
6. Recibe una cadena de caracteres y muestra el número de letras repetidas por ejemplo: Ana A se presenta 2 veces N se presenta 1 vez Prototipo:		
7. Recibe un arreglo de enteros y devuelve el dato mayor. Prototipo:		
8. Recibe un arreglo de cadena de caracteres, la función solicita un nombre muestra la posición del nombre en el arreglo si lo encuentra y si no informa que no se encuentra dicho nombre. Prototipo:		

2. Codifica las siguientes funciones:

1. Escribir una función que lea los tres lados y devuelva la superficie del triángulo con base en la formula $A = \sqrt{p \times (p-a) \times (p-b) \times (p-c)}$ Donde $p = (a+b+c)/2$

Codificación	Invocación de la función

2. Codifique una función que lea 2 números enteros; la función no recibe parámetros y devuelve el producto de los dos valores leídos.

Codificación	Invocación de la función

3. Codifique una función que lea 2 números enteros que divida el primero entre el segundo y lo muestre en pantalla (la función no recibe parámetros)

Codificación	Invocación de la función

4. Codifique una función que determine el área de un triángulo (la función recibe la base y la altura como parámetros)

Codificación	Invocación de la función

5. Codifique una función que determine el área de un circulo (la función recibe el radio como parámetro).

Codificación	Invocación de la función

6. Codifique una función que reciba un número entero y nos indique si es negativo o positivo mediante un mensaje en la pantalla

Codificación	Invocación de la función

7. Codifique una función que determine en un numero entero si es par o impar, la función recibe el número como parámetro.

Codificación	Invocación de la función

8. Codifique una función que lea del teclado 3 números enteros positivos diferentes y devuelva el valor mayor

Codificación	Invocación de la función

9. Codifique una función que lea 3 número enteros positivos diferentes y devuelva el menor.

Codificación	Invocación de la función

10. Codifique una función que lea un número entre 1 y 9 y que muestre en pantalla los primeros 10 múltiplos del número leído.

Codificación	Invocación de la función

11. Codifique una función que lea un número de 3 dígitos y que lo muestre con las posiciones invertidas ejemplo: se lee 529 y se muestra 925

Codificación	Invocación de la función

12. Codifique una función que reciba una palabra y se muestre invertida, ejemplo: le lee “Teresa” y se muestra “asereT”.

Codificación	Invocación de la función

13. Codifique una función que reciba como parámetro un número entero positivo menor a 100 y que muestre en pantalla los números primos incluidos en el rango del 1 al número recibido

Codificación	Invocación de la función

14. Codifique una función que determine si un número es capicúa (es un número que es igual al derecho y al revés, ejemplo: 12321

Codificación	Invocación de la función

15. Codifique una función que genere los primeros tres números perfectos (Un número perfecto es un número natural que es igual a la suma de sus divisores propios positivos. Así, 6 es un número perfecto porque sus divisores propios son 1, 2 y 3; y 6 = 1 + 2 + 3. Los siguientes números perfectos son 28, 496 y 8128)

Codificación	Invocación de la función

16. Codifique una función que reciba un número entero entre 1 y 3999 y lo escriba en la pantalla en notación romana.

Codificación	Invocación de la función

17. Codifique una función que lea un número e indique si fue año bisiesto o no (Un año es bisiesto en el calendario Gregoriano, si es divisible entre 4, excepto aquellos divisibles entre 100 pero no entre 400).

Codificación	Invocación de la función

18. Codifique una función que lea un número y escriba su nombre en español

Codificación	Invocación de la función

19. Codifique una función que sume del 1 al 500, solo impares

Codificación	Invocación de la función

20. Codifique una función que lea 15 números y que indique cuantos son negativos y cuántos son positivos

Codificación	Invocación de la función

21. Codifique una función que lea 10 números positivos y obtener el promedio, indicar cuál fue el menor y cual el mayor

Codificación	Invocación de la función

22. Codifique una función que lea un número y lo sume

Codificación	Invocación de la función

23. Codifique una función que reciba los coeficientes de una ecuación de segundo grado y aplicando la formula general muestre el valor de X

Codificación	Invocación de la función

24. Codifique una función que determine la suma de números al cuadrado de 1 hasta n, si n es un numero entero positivo menor a 500.

Codificación	Invocación de la función

25. Codifique una función que muestre todas las tablas de multiplicar del 2 al 9

Codificación	Invocación de la función

26. Codifique una función que visualizar los primeros n números al cuadrado; donde n es un número introducido por el usuario y que muestre también la suma de todos ellos.

Codificación	Invocación de la función

27. Codifique una función que genere 50 números aleatorios en "C"(la función no recibe ni devuelve valores)

Codificación	Invocación de la función

28. Codifique una función que genere 10 números aleatorios entre 0 y 9 y que los devuelva en una estructura tipo arreglo

Codificación	Invocación de la función

29. Codifique una función que visualice un triángulo con asteriscos, el usuario introduce un número y ese número será el número de asteriscos en la base

Codificación	Invocación de la función

30. Codifique una función que devuelva el cuadrado de la suma de tres valores recibidos como parámetros.

Codificación	Invocación de la función

31. Codifique una función que devuelva la suma de dos números enteros

Codificación	Invocación de la función

32. Codifique una función que escriba n veces "me gusta programar en c", n es un número recibido como parámetro.

Codificación	Invocación de la función

33. Codifique una función dibuje n veces un cuadrado de asteriscos; el número n y el tamaño del cuadrado serán recibidos como parámetros de tipo entero.

Codificación	Invocación de la función

34. Codifique una función que muestre un menú 1) el cuadrado de un número entero. 2) la suma de dos números enteros. 3) el área de un cuadrado y que invoque a las funciones necesarias para su funcionamiento. Considere todas las funciones necesarias.

Codificación	Invocación de la función

35. Codifique una función que mediante un menú realice las operaciones necesarias para: 1) obtener la factorial de un numero; 2) obtener los divisores de un numero; 3) los números impares de un numero introducido por el usuario. Codifique todas las funciones necesarias

Codificación	Invocación de la función

Capítulo 5
Registros

5.1 Registros

Así como los arrays son organizaciones secuenciales de variables simples, de un mismo tipo cualquiera dado, resulta necesario en múltiples aplicaciones, agrupar variables de distintos tipos, en una sola entidad.

Este sería el caso, si quisiéramos generar la variable "datos_personal " , en ella tendríamos que incluir variables del tipo: strings , para el nombre , apellido , nombre de la calle en donde vive , etc , enteros , para la edad , número de código postal , float (ó double , si tiene la suerte de ganar mucho) para el sueldo , y así siguiendo.

Existe en C en tipo de variable compuesta, para manejar esta situación típica de datos, llamada ESTRUCTURA. No hay limitaciones en el tipo ni cantidad de variables que pueda contener una estructura, mientras su máquina posea memoria suficiente como para alojarla, con una sola salvedad: **una estructura no puede contenerse a sí misma como miembro**.

Para usarlas, se deben seguir dos pasos. Hay que, primero declarar la estructura en sí, esto es, darle un nombre y describir a sus miembros, para finalmente declarar a una ó más variables, del tipo de la estructura antedicha, Ejemplo:

```
struct empleado{
                        int edad ;
                        char nombre[50] ;
                        float sueldo ;          } ;

struct empleado  vendedores, profesionales;
```

En la primera sentencia se crea un tipo de estructura, mediante el declarador "struct", luego se le da un nombre "empleado" y finalmente, entre llaves se declaran cada uno de sus miembros, pudiendo estos ser de cualquier tipo de variable, incluyendo a los arrays ó aún otra estructura.

La única restricción es que no haya dos miembros con el mismo nombre, aunque sí pueden coincidir con el nombre de otra variable simple, (o de un miembro de otra estructura), declaradas en otro lugar del programa. **Esta sentencia es sólo una declaración, es decir que no asigna lugar en la memoria para la estructura, sólo le avisa al compilador como tendrá que manejar a dicha memoria para alojar variables del tipo struct empleado**.

En la segunda sentencia, se definen dos variables del tipo de la estructura anterior, (ésta definición debe colocarse luego de la declaración), y se reserva memoria para ambas.

Las dos sentencias pueden combinarse en una sola, dando la definición a continuación de la declaración:

```
struct empleado{
            int edad ;
            char nombre[50] ;
            float sueldo ;
          }   vendedor, programador ;
```

Y si no fueran a realizarse más declaraciones de variables de este tipo, podría obviarse el nombre de la estructura (empleado).

Las variables del tipo de una estructura pueden ser inicializadas en su definición, así por ejemplo se podría escribir:

```
struct empleado{
            int edad ;
            char nombre[50] ;
            float sueldo ;
            char observaciones[500] ;
          }   vendedor = { 40 , "Juan Eneene" , 1200.50 , "Asignado a zona A"  };
```

en el siguiente ejemplo se utilizaron las dos modalidades de definición de variables, inicializándolas a ambas.

```
struct empleado  programador = {23, "José Pérez", 2000.0, "Asignado a zona B"};
```

5.2 REGLAS PARA EL USO DE ESTRUCTURAS

Lo primero que debemos estudiar es el método para dirigirnos a un miembro particular de una estructura. Para ello existe un operador que relaciona al nombre de ella con el de un miembro, este operador se representa con el punto (**.**) , así se podrá referenciar a cada uno de los miembros como variables individuales , con las particularidades que les

otorgan sus propias declaraciones, internas a la estructura. La sintaxis para realizar esta referencia es:

nombre_de_la_estructura.nombre_del_miembro, así podremos escribir por ejemplo, las siguientes sentencias

```
strut posicion_de {
        float eje_x ;
        float eje_y ;
        float eje_z ;
        } fin_recta , inicio_recta = { 1.0 , 2.0 , 3.0 ) ;

fin_recta.eje_x = 10.0;
        fin_recta.eje_y = 50.0;
        fin_recta.eje_z = 90.0;
```

```
if( fin_recta.eje_x == inicio_recta.eje_x )
```

Es muy importante recalcar que, dos estructuras, aunque sean del mismo tipo, no pueden ser asignadas o comparadas la una con la otra, en forma directa, sino asignando ó comparándolas miembro a miembro. Esto se ve claramente explicitado en las líneas siguientes, basadas en las declaraciones anteriores:

```
fin_recta = inicio_recta;           /* ERROR */
if ( fin_recta >= inicio_recta );      /* ERROR */
fin_recta.eje_x = inicio_recta.eje_x ;    /* FORMA CORRECTA DE ASIGNAR */
fin_recta.eje_y = inicio_recta.eje_y ;    /* UNA ESTRUCTURA A OTRA     */
fin_recta.eje_z = inicio_recta.eje_z ;

if( (fin_recta.eje_x >= inicio_recta.eje_x) &&     /* FORMA CORRECTA DE  */
(fin_recta.eje_y >= inicio_recta.eje_y) &&     /* COMPARAR UNA       */
(fin_recta.eje_z >= inicio_recta.eje_z) )      /* ESTRUCTURA CON OTRA */
```

Las estructuras pueden anidarse, es decir que una o más de ellas pueden ser miembro de otra. Las estructuras también pueden ser pasadas a las funciones como parámetros, y ser retornadas por éstas, como resultados.

5.3 ARRAYS DE ESTRUCTURAS

Los arrays se pueden agrupar, para formarlos, cualquier tipo de variables, esto es extensible a las estructuras y podemos entonces agruparlas ordenadamente, como elementos de un array.

Ejemplo:

```
typedef struct {
            char     material[50] ;
            int       existencia ;
            double costo_unitario ;
           } producto;

producto  stock[100];
```

Hemos definido aquí un array de 100 elementos, donde cada uno de ellos es una estructura del tipo producto compuesta por tres variables, un int , un double y un string ó array de 50 caracteres.

Los arrays de estructuras pueden inicializarse de la manera habitual, así en una definición de stock, podríamos haber escrito:

```
producto   stock1[100] = {
                "tornillos"  , 120 , .15 ,
                "tuercas"    , 200 , .09 ,
                "arandelas"  ,  90 , .01     } ;

producto    stock2[ ]     = {
                 {'i','t','e','m','1','\0' } , 10 , 1.5 ,
                 {'i','t','e','m','2','\0' } , 20 , 1.0 ,
                 {'i','t','e','m','3','\0' } , 60 , 2.5 ,
                 {'i','t','e','m','4','\0' } , 40 , 4.6 ,
                 {'i','t','e','m','5','\0' } , 10 , 1.2 , } ;
```

5.4 CUESTIONARIOS Y EJERCICIOS

1. Codifique un programa que permita almacenar un grupo de registros con información de 30 prendas de vestir, se debe almacenar la talla, la cantidad almacenada, el modelo, el

color, el nombre de la prenda y el precio. El programa debe mostrar un menú de trabajo que permita mostrar los datos de las prendas en forma de tabla, almacenar prendas, modificar datos de prendas, eliminar los datos de una prenda (por eliminar se entiende guardar datos en cero para los datos numéricos y cadena vacía para los campos tipo string). El usuario decide la posición en la que se almacena los datos de la prenda y la salida del programa. Use funciones.

2. Explique cada línea de código

Código	Significado en la ejecución
1. #include <stdio.h>	
2. #include <string.h>	
3. #define MAX 2	
4. #define TAMPALABRA 50	
5. struct curso {	
6. char nomCurso[TAMPALABRA];	
7. char nomProfesor[TAMPALABRA];	
8. int hrs;	
9. float costo;	
10. };	
11. main(){	
12. struct curso micurso[MAX];	
13. char nuNombre[TAMPALABRA];	
14. char nuProf[TAMPALABRA];	
15. int i;	
16. for(i=0;i<MAX;i++){	
17. printf ("Introduce el nombre del curso \t");	
18. scanf("%[^\n]s",&micurso[i].nomCurso); getchar();	
19. printf ("Introduce el nombre del profesor \t");	
20. scanf("%[^\n]s",&micurso[i].nomProfesor);	
21. printf ("Introduce el costo del curso\t");	
22. scanf("%f",&micurso[i].costo);	
23. printf ("Introduce la duración del curso \t");	
24. scanf("%d",&micurso[i].hrs);getchar();	
25. }	
26. printf("\n \n Datos guardados\n");	
27. printf ("Curso \t Profesor: \t Costo del curso:\t HRS del curso: \n");	
28. for (i=0;i<MAX;i++){	
29. printf("%s \t %s \t %.2f \t %d \n",	

micurso[i].nomCurso,micurso[i]. nomProfesor,micurso[i].costo,micurso[i].hrs); }	
30. getchar(); getchar();	
31. /*	
32. printf ("Introduce un nuevo nombre del curso \t");	
33. scanf("%[^\n]s",&nuNombre);getchar();	
34. printf ("Introduce un nuevo profesor \t");	
35. scanf("%[^\n]s",&nuProf);getchar();	
36. strcpy(micurso[0].nomCurso,nuNombre);	
37. strcpy(micurso[0].nomProfesor,nuProf);	
38. for (i=0;i<MAX;i++){	
39.printf("%s \t %s \t %.2f \t %d \n",micurso[i]. nomCurso,micurso[i].nomProfesor,micurso[i]. costo,micurso[i].hrs); } */ }	

1. Una inmobiliaria tiene información sobre departamentos en renta, de cada departamento se conoce:

   - Clave: es un entero que identifica al inmueble
   - Extensión: superficie del departamento en metros cuadrados
   - Ubicación: (excelente, buena, regular, mala)
   - Precio: es un valor real.
   - Disponible: VERDADERO se ésta disponible para la renta y FALSO si ya está rentado.

Diariamente acuden muchos clientes a la inmobiliaria solicitando información. Escriba un programa capaz de realizar las siguientes operaciones sobre la información disponible:

a) Liste los datos de todos los departamentos disponibles que tengan un precio inferior o igual a cierto valor P

b) Liste los datos de los departamentos disponibles que tengan una superficie mayor o igual a cierto valor E y una ubicación excelente

c) Liste el monto de la renta de todos los departamentos alquilados

d) Llega un cliente solicitando un departamento, se listen los departamentos que se ajusten a las posibilidades del cliente (precio, extensión, ubicación), si se encuentra uno que se ajuste se registre como alquilado y se almacenen los cambios

Bibliografía

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

2. Domínguez Vera Edgar Danilo, 2014, “Programación Estructurada: Raptor y lenguaje C”, AlfaOmega, México

3. Cairó Osvaldo, 2014, "Fundamentos de Programación Piensa en C", Prentice Hall, México.

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

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

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

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

8. Vázquez Peña Mario "Introducción al lenguaje C", Universidad Autónoma Chapingo, México 1997