Problemario para CI-2125: Computación I
(versión revisada)

Rosseline Rodríguez

Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar

REPORTE CI-013-96
Febrero 2000

Resumen

En el presente reporte se presenta una selección de problemas para el curso Computación I que abarcan los tópicos principales dictados en este curso. Esta selección está basada en un primer problemario (Reporte CI-013-96) para CI-2125, el cual ha sido revisado y ampliado en cada uno de sus tópicos. Adiconalmente se ha agregado un capítulo que incluye los tópicos Especificaciones de Entrada y Salida, Algoritmos y Programación Estructurada, los cuales forman parte del curso, pero no fueron considerados en el problemario anterior.

Cada capítulo del problemario se ha estructurado en dos secciones. Una sección inicial de ejemplos para ilustrar situaciones comunes presentadas en la mayoría de los programas y una sección de ejercicios dejados planteados para el estudiante. Se espera que este problemario sea de apoyo a las horas de práctica del estudiante.

Entendemos que el material presentado está sujeto a correcciones y ampliaciones futuras las cuales gustosamente haremos con el valioso aporte que el lector crítico pueda hacer del mismo; por lo que esperamos de estudiantes, preparadores y profesores que nos pongan al tanto de sus observaciones y sugerencias.

Apartado Postal 89000, Caracas 1080-A - Teléfono: 906-3269 Fax: 906-3243
crodrig@usb.ve leonid@usb.ve

Índice

CAPÍTULO 1

Especificaciones de Entrada y Salida, Algoritmos, y Análisis Descendente

Ejemplos:

1.- La mayoría de los problemas que aparecen en los libros o se le presentan a los estudiantes durante la carrera están bien definidos y son precisos. En la vida real, los problemas rara vez se presentan sin ambigüedades. Como es el caso del siguiente enunciado:

"Dado un texto, clasificar las palabras según su largo
y calcular el porcentaje de cada tipo."

2.- Este tipo de enunciados nos obligan a interactuar con la persona que propuso el problema, para producir un conjunto de especificaciones claras, no ambiguas que llevan a aclarar lo que deseamos hacer. Estas especificaciones se conocen como **especificaciones de entrada y salida, y procesamiento especial**, y se obtienen a traves de preguntas que se hacen a la persona que planteó el problema.

3.- Si vemos el plantemiento del numeral 1, para la entrada, algunas preguntas pertinentes serian: ¿los valores de entrada son caracteres?, en ese caso, ¿qué caracteres podemos encontrar?, ¿cómo saber cuando se acabó el texto?, ¿qué pasa con la puntuación?

4.- Al interactuar con el futuro usuario del programa podemos concluir como especificación de entrada lo siguiente:*"La entrada está constituida por caracteres, que pueden ser letras (mayúsculas o minúsculas), espacios en blanco y los signos de puntuación ('.', ','). El texto siempre finaliza con el caracter '$'. "*

5.- En cuanto a la salida, algunas preguntas pertinentes serian: ¿qué mensajes producimos para cada tipo de palabra?, ¿hay que imprimir información adicional? ¿algún formato especial?, ¿qué medio se utilizará, pantalla o impresora? Cuya conclusión al final de las preguntas es: *La salida debe estar aparecer en pantalla de la siguiente forma:*

Tipo de Palabra	*Porcentaje*
Corta	*%*
Media	*%*
Larga	*%*
ExtraLarga	*%*

6.- En cuanto al procesamiento especial, algunas preguntas pertinentes serian: ¿las palabras puedes estar separadas por más de un espacio en blanco?, ¿la entrada puede ser vacía?, ¿los signos de puntuación se consideran parte de la palabra o simplemente separadores?, ¿qué es una palabra corta, media, larga, extralarga?. Finalmente concluimos: *Las palabras pueden venir separadas por más de un espacio en blanco. La entrada puede ser vacía, es decir, sólo el signo '$' aparece en ella. Los signos de puntuación son separadores de palabras*

que siempre siguen al último caracter de una palabra y pueden editar seguidos de un blanco o un '$'. Las palabras se clasifican así: corta entre 1 y 3 caracteres, media entre 4 y 6 caracteres, larga entre 7 y 9 caracteres, extralarga de 10 caracteres en adelante.

7.- Un **algoritmo** es una secuencia ordenada de operaciones bien definidas y efectivas, que al ser ejecutada, siempre produce un resultado, y siempre termina en un tiempo finito. Veamos el siguiente ejemplo, de una secuencia de pasos que indica cómo lavarse el cabello y analicemos si es un algoritmo.

1. *Mojar el cabello*
2. *Echar champú*
3. *Hacer espuma*
4. *Enjuagar*
5. *Repetir*

El paso 5 produce ambigüedad pues no dice desde dónde repetir, no dice cuál es el próximo paso a la instrucción 5. Como conclusión, un algoritmo es un procedimiento para ejecutar una tarea particular, cuyas características son: identidad del próximo paso, punto de comienzo y puntos finales son claramente definidos, termina en un numero finito de pasos, y compuesto de primitivas no ambiguas.

8.- Las formas más comunes de especificar los algoritmos son el **Lenguaje Algorítmico o formalismo, el lenguaje natural y los Diagramas de Flujo.** Los **formalismos** representan un compromiso ideal entre dos extremos, el lenguaje natural y el lenguaje de programación. El lenguaje natural es engorroso para especificar algoritmos, no está limitado a un vocabulario, ni tiene ninguna relación directa con un lenguaje de programación. Los **Diagramas de Flujo** constituyen un gráfico de la solución de un problema. Está formado por un conjunto de cajas que indican el tipo de la operación a ejecutar.

9.- La programación estructurada también conocida como análisis descendente, técnica de programación TOP-DOWN, desarrollo modular o refinamiento, inicialmente describe el problema en el más alto nivel: ¿ qué debemos hacer y no cómo hacerlo ? El algoritmo no se describe en base a primitivas sino en base a operaciones de alto nivel. Cada una de estas operaciones es refinada a un segundo nivel, continuando este proceso de refinamiento hasta llegar a instrucciones primitivas.

10.- Tomando el problema del numeral 6, donde se clasifican las palabras de un texto según su tipo, se puede plantear el siguiente algoritmo de alto nivel;

```
INICIO
    CICLAR
        SI el texto se acabó
            TERMINAR
        Encontrar la próxima palabra
    Clasificar la palabra
    Calcular el porcentaje de cada tipo
    ESCRIBIR resultados
FIN
```

11.- A continuación se analizarán aquellas partes del algoritmo que deben refinarse, esas partes aparecen en cursivas.

SI *el texto se acabo*: es necesario leer el próximo caracter para reconocer si es el caracter '$' en ese caso se terminó el análisis del texto. Note que la primera vez que se entra al ciclo, se está chequeando que el texto no venga vacío, es decir, que el primer y único caracter que aparece es '$'. El algoritmo quedaría

```
TOMAR caracter
SI caracter == FIN
   TERMINAR
```

Encontrar la próxima palabra: primero se saltan todos los espacios en blanco encontrados, pues es factible tener varios espacios seguidos. Al encontrar un caracter que no espacio en blanco estamos seguros que encontramos una palabra por lo cual simplemente contamos los caracteres que tiene hasta que aparezca algún separados de palabras (espacio en blanco, coma o punto) en el texto. Note que se debe pasar al próximo caracter una vez que encontramos el final de la palabra, pues este puede ser un punto o una coma, es decir un caracter distinto a espacio en blanco. En este caso el algoritmo correspondiente a encontrar próxima palabra seria:

```
CICLAR
    SI caracter >< BLANCO
        TERMINAR
    TOMAR caracter
    largo←1
    CICLAR
        SI (caracter == BLANCO)
         O (caracter == PUNTO)
         O (caracter == COMA)
            TOMAR caracter
            TERMINAR
        largo←largo+1
        TOMAR caracter
```

Clasificar la palabra: De acuerdo al largo calculado para la palabra actual es sencillo clasificarla solo hay que preguntarnos a que rango pertenece, como se muestra a continuación:

```
SI largo < 3
    corta←corta+1
SINO SI largo < 6
    media←media+1
SINO SI largo ≤ 9
    larga←larga+1
```

```
SINO
    extra←extra+1
```

Calcular porcentaje de cada tipo: los dos siguientes trozos del algoritmo en realidad ejecutan esta instrucción de alto nivel. En el primer trozo se calcula el total de palabras obtenido para después calcular el porcentaje. En caso de que el total es cero, no hay palabras en el texto, se indica con una 'N'.

```
total←corta+media+larga+extra
SI total == 0
   ESCRIBIR 'N'/*No hay palabras en el texto*/
```

Escribir resultados: en caso de que el total es diferente de cero, se procede a escribir el reporte obtenido, con sus porcentajes respectivos.

```
SINO
    ESCRIBIR 'Tipo de Palabra Promedio'

    ESCRIBIR 'CORTA ', corta/total*100
    ESCRIBIR 'MEDIA ', media/total*100
    ESCRIBIR 'LARGA ', larga/total*100
    ESCRIBIR 'EXTRALARGA ', extra/total*100
```

Ejercicios:

1.- Para los siguientes enunciados de problemas, determine las especificaciones de entrada, especificaciones de salida y casos especiales que pueden aclarar dicho enunciado.

a) Realice un programa que lleve el inventario de la sala de lectura de Ingeniería Electrónica
b) Se quiere que usted haga un programa que simule el comportamiento de un circuito digital.
c) Una de las actividades que realiza la Coordinación de Ingeniería Electrónica, es llevar el archivo de los libros de tesis y pasantías larga que son parte de los requisitos para optar al título de Ingeniero Electrónico. Además de llevar un control, se provee a los estudiantes de esta carrera de una biblioteca que puede ser consultada con fienes académicos. Se desea que usted realice un programa que permita controlar la entrada de libros y los préstamos de los mismos.
d) Un estudiante de Matemáticas necesita hacer un programa que grafique funciones, y le pide a usted ayuda para diseñar el sistema.
e) Usted es un fanático de beisbol y desea llevar un registro de todos los eventos ocurridos en las finales de la liga nacional e internacional. Para ello se plantea hacer un programa que simplifique este registro.
f) Escribir un programa que haga la asignación de salones a materias y secciones de DACE.
g) Hacer un programa que haga efectiva la inscripción de los estudiantes de la USB.
h) Un amigo suyo, estudiante de Ingeniería Química, debe hacer programas que resuelven Métodos Numéricos en calculadoras HP. Le a pedido ayuda para conectar su

calculadora a un impresora laser, con el fin de poder imprimir los resultados de las corridas de sus programas. Usted decide hacerlo usando como puente su computador personal. Diseñe un programa que tome los datos de la calculadora vía puerto paralelo y los envíe a la impresora.

2.- Diseñe un algoritmo para resolver cada uno de los siguientes problemas:

a) Dados dos números naturales, decidir si uno es divisor del otro
b) Calcular el cociente y el resto de la división de dos naturales
c) Decidir si una palabra es palíndrome
d) Verificar si una fracción es irreducible
e) Hallar la solución de una ecuación de segundo grado
f) Validar una fecha en formato DD/MM/AA
g) Determinar la edad de una persona
h) Invertir un número entero de a lo sumo diez cifras
i) Determinar la distribución óptima en billetes y monedas para una suma de dinero

3.- Para cada una de las siguientes situaciones de la vida real, decir si puede o no resolverse con un algoritmo, en caso afirmativo, diseñar el algoritmo:

a) Preparar una torta
b) Indicarle a una persona cómo llegar a una dirección dada
c) Jugar ajedrez
d) Llamar por teléfono
e) Traducir un texto de Inglés a Español

4.- El método que Ud. aprendió en la educación básica para realizar la división. ¿es un algoritmo? ¿por qué?

5.- Dadas las siguientes secuencias de pasos, decir si son o no algoritmos (justifique). En caso de que no sean indique si se pueden modificar para obtener un algoritmo y realice los cambios pertinentes.

a) Para hornear una torta realice los siguientes pasos:
   i) Prenda el Horno a 350°
   ii) Meta la torta en el horno
   iii) Espere hasta que esté cocida
   iv) Saque la torta del horno

b) Para hacer arepas:
   i) En un recipiente limpio, eche una taza de harina
   ii) Por cada taza de harina, eche taza y media de agua tibia
   iii) Eche sal al gusto
   iv) Amase bien hasta que todo quede homogéneo
   v) Si la masa queda dura, agregar más agua
   vi) Realice con la masa las arepas

c) Si se encuentra en la estación Chacaíto y quiere ir a La Hoyada:
   i) Diríjase a la caseta del operador
   ii) Cuando le toque su turno, pida un boleto hasta La Hoyada

iii) Entregue dinero suficiente (>= que la cantidad pedida por el operador)
iv) Tome el boleto y diríjase al torniquete
v) Introduzca el boleto en el torniquete y pase a través de la barra
vi) Diríjase al andén de trenes con dirección PROPATRIA
vii) Cuando llegue el tren y abra sus puertas, suba, una vez que termine la salida de personas
viii) Durante el recorrido esté pendiente de los nombre de las estaciones
ix) Cuando se detenga en la estación La Hoyada, salga del tren al abrir la puerta
x) Diríjase a los torniquetes
xi) Introduzca su boleto en el torniquete y pase a través de la barra

6.- Hacer un algoritmo para calcular el cociente y el resto de la división de números naturales.

7.- Usando la técnica del análisis descendende escriba algoritmos para los siguientes problemas. Utilice como formas de representación diagramas de flujo y pseudocódigo

a) Calcular el cociente y el resto de la división de números naturales
b) Leer tres números naturales y calcular el máximo el mínimo y el promedio de esos números
c) Dada una temperatura en grados Fahrenheit, devolver su equivalente en grados celsius.
d) Dado un texto terminado en $ diga cuantas veces aparece cada una de las vocales.
e) Dados tres valores enteros escribirlos en forma ascendente.
f) Dada una lista de enteros positivos terminada en -1, encontrar la suma de ellos, el total de números leidos y el promedio
g) Dado el lado de un cuadrado calcular el perímetro y el área.
h) Dados dos vectores A y B calcular su suma de ellos
i) Dado un vector V calcular su norma $\sqrt{v_1^2 + ... + v_n^2}$
j) Dada una matriz M decir si es diagonal
k) Dadas dos matrices M y N calcular la suma M+N
l) Calcular el índice académico de un trimestre
m) Calcular las raices de una ecuación cuadrada Ax^2+Bx+C

8.- Para los siguientes algoritmos, escriba una tabla de trazas con los valores por los que pasan las variables:

a)

```
INICIO
(1)   X←0, Y←5, Z←25
      MIENTRAS X≤4 HACER
(2)      Y←Z-Y, A←X+1, X←X+1
         SI A>1
(3)         Z←Z-5, A←A^2, B←Z-Y
      FIN DE MIENTRAS
(4)   MOSTRAR A,B,X,Y,Z
FIN
```

Tabla de Trazas:

Instrucción	A	B	X	Y	Z
1	?	?	0	5	25
2	1	?	1	20	25
2	2	?	2	20	25
3	4	0	2	20	20
...					

b) Suponga que los valores introducidos para A son 0.1, 0.3, 1.0

```
    INICIO
(1)    LEER A
       MIENTRAS A≤0.3 HACER
(2)       A←A+1, X←X+1
          SI A≠0.3
(3)          S←0, X←0, T←1
             MIENTRAS T≤6 HACER
(4)             T←T+X, T←T+2
          FIN DE SI
(5)       MOSTRAR A,S,X
       FIN DE MIENTRAS
    FIN
```

c) Suponga que los conjuntos de valores para b,h,k son 3,6,1 luego 4,3,2; luego 5,2,0 y finalmente 2,6,2

```
    INICIO
(1)    I,A,X←0
       MIENTRAS X≤4 HACER
(2)       I←I+1
(3)       LEER b,h,k
          SI k≥1
(4)          A←(bh)/2
             SI K≥2
(5)             X←(bh^3)/36
          FIN DE SI
(6)       MOSTRAR I,b,h,A,X
       FIN DE MIENTRAS
    FIN
```

CAPÍTULO 2

Identificadores y Evaluación de Expresiones
Definiciones de Tipos Simples

Ejemplos:

1.- **`Suma`** es un identificador permitido en C pues contiene caracteres comprendidos en el conjunto {a..z}∪{A..Z}∪{0..9}∪{_} y no comienza por un dígito.

2.- **`while`** no es un identificador permitido ya que es una palabra reservada del lenguaje C.

3.- **`True`** puede ser el identificador de una constante en un programa en C si aparece una definición como la siguiente al comienzo del archivo:

```
#define True 1
```

y puede ser usado con semántica de valor verdadero.

4.- En la siguiente declaración de constantes

```
#define 2 'c'
```

decimos que la declaración es *errónea sintácticamente* ya que **`2`** no cumple con las reglas sintácticas de C que definen un identificador.

5.- Una declaración de constantes *sintácticamente correcta* es la siguiente:

```
#define CAR 'x'
```

ya que **`CAR`** define un identificador correcto en C al cual se le asocia el caracter 'x'.
Los tipos simples provistos por C son **`int`** (para el conjunto de los enteros), **`char`** (para el conjunto de los caracteres de la máquina) y **`float`** (para el conjunto de los reales).

6.- La expresión `1+2*3` es equivalente a evaluar `1+(2*3)`, es decir, la evaluación resultante es el entero `7`; lo cual es diferente a evaluar `(1+2)*3` cuyo valor es `9`. De lo anterior podemos concluir que los paréntesis pueden ayudar a entender y/o cambiar la semántica de las expresiones.

7.- La expresión **`0||1`** tiene valor **verdadero**. La expresión **`(0||1)&&1`** resulta en **verdadero**, que no es igual a evaluar **`0||(1&&0)`** la cual resulta en **`0`** o **falso**.

8.- Para declarar una variable, sólo es necesario escribir el tipo seguido de los identificadores:

<tipo> <lista de identificadores>;

9.- Además de los tipos simples predefinidos provistos por C, se pueden definir nuevos tipos a través de la instrucción **`typedef`**, la cual se localiza en el encabezado del programa antes de las declaraciones de variables que usen dicho tipo. Para definir nuevos tipos debe escribirse:

`typedef` <definición> <nombre_del_tipo>;
...

`typedef <definición> <nombre_del_tipo>;`

Si queremos definir el tipo **boolean**, para los valores de la verdad cero (**falso)** y distinto de cero (**verdadero),** colocamos la instrucción

`typedef int boolean;`

Ejercicios:

1.- ¿Qué errores existen en las siguientes declaraciones?

```
a) #define mitad .5;
b) #define 0 falso
c) float c, int d;
d) int=x;
e) char numero; real letra;
f) int x,y,z,t,n,v; float x; char n,m,f,g,
g) #define n 12;
h) #define 1 'a', z 5; 15 z;
```

2.- ¿ Cuáles son los valores de la siguientes expresiones?

```
a) 3.0*5.0+3.0/2.0+7.0
b) 3.0*(5.0+3.0)/2.0+7.0
c) 3.0*5.0+(3.0/2.0)+7.0
d) 3.0+(5.0+3.0/2.0)+7.0
e) 4 % 2+6 / (2*3)
f) 4 % (2+6 / 2)*3
g) (4 % 2+6) / 2*3
h) 4 % 2 / 2*3
i) 12*12
j) 144/10
k) 144 / 10
l) 144 % 10
m) 1 && 0
n) 12!=22 || 12<12
o) 123+456
p) (12!=22)<(12>0.5)
q) 0 || 1
r) 2* (2%4)
```

3.- Indique si los siguientes identificadores son o no válidos, de no serlo explique el motivo:

```
a) Pi
b) 23Pi
c) UpCase
d) Seno_hiperbolico_de_93_grados
e) _Pepito
f) Elem32
```

4.- ¿Cuál de las siguientes expresiones son válidas en C?. Hallar el valor de aquellas que sean válidas.

```
a) a*b/2(1.0-a)
b) (x+y)/4.0+1)
c) ((c1)+1))
d) && b=1 || 5
e) ('b' = 66) || (66 = 'B')
f) ((1 && (4>2)) || !(65 != 'A')) || (1 && (!(!(!(4>=5 && 0)))))
g) (1 && (!(!(0)) || (6>8)) && (!(4<9) || 1)
```

5.- Usando las reglas de precedencia de operadores de C, calcule el valor de las siguientes expresiones. En cada caso indique el orden de cálculo de los operadores:

```
a) 3+2*7/4
b) 15+9/3+5
c) 2*5*5+3*5+7
d) (15+9)/(3+5)
e) (2+3*5)/3
f) (4*7-3*5)%5
g) A==B&C==D
h) x=7+3*6/2-1
i) x
j) (A==1)&(B==2)||(C==3)
k) 3*4%7+2/5-3
```

6.- Dada la ecuación $y=ax^3 + 7$ ¿cuál de las que siguen, si existe alguna, es una expresión correcta en C correspondiente a esa ecuación?

```
a) y = a*x*x*x + 7;
b) y = a*x*x*(x + 7);
c) y = (a*x)*x*(x + 7);
d) y = (a*x)*x*x + 7;
e) y = a*(x*x*x) + 7;
f) y = a*x*(x*x + 7);
g) y = (a*(x*x*x)) + 7;
```

7.- Diga cuáles de los siguientes enunciados son verdaderos y cuáles son falsos. Explique sus respuestas.

a) Los operadores C se evalúan de izquierda a derecha.
b) Los siguientes son todos nombres válidos de variables: `_under_bar_, m928134, t5, j7, her_sales, his_account_total, a, b, c, z, z2.`
c) Una expresión aritmética en C que no contenga paréntesis se evalúa de izquierda a derecha.
d) Los siguientes son todos nombres inválidos de variables: `3g, 87, h22, 2h, 67h2.`

8.- Declare el orden de cálculo de los operadores en cada una de las siguientes expresiones de C, y muestre el valor de `x` después de que se ejecute cada uno de ellos.

```
a) x = 7 +3 *6 / 2 -1;
b) x = 2 % 2 + 2 * 2 - 2 / 2;
c) x = (3 * 9 * (3 + (9 * 3 / (3))));
```

9.- Dé definiciones de constantes y de tipos usando `#define` y los tipos `int`,`char` y `float` para:

a) Colores primarios
b) Días de la semana
c) Vocales y consonantes
d) Meses del año
e) Mayúsculas y minúsculas
f) Minoría de Edad

Capítulo 3

Instrucciones Simples, Condicionales y Ciclos

Ejemplos:

1.- El siguiente programa calcula X^2 si X>0 y $|X|$ si X<0, para ello hace uso de un **condicional** .

```
#include <stdio.h>
main() {
     int X,resp;

     scanf("%d",&X);
     resp=X;
     if (X>0)
          resp=X*X
     else
          resp=X*(-1);
     printf("%d",resp);
}
```

La instrucción **`scanf`** permite la lectura de valores desde el teclado (entrada estándar). El parámetro `"%d"` indica que lo leído es un valor entero y `&X` indica que la variable `X` será modificada con el nuevo valor leído. La instrucción **`printf`** permite escribir valor en la salida estándar o pantalla, y al igual que con scanf "`%d`" indica que lo que se escribe debe ser un entero. La variables **`resp`** no se modifica al escribir por lo cual no va antecedida de un amperson (`&`). Para usar ambas instrucciones (`scanf` y `printf`) es necesario que en el encabezado del programa se coloque la directiva de compilación **`"#include <stdio.h>"`**. Analizando el programa se observa que la inicialización de la variable **`resp`** antes del condicional es redundante, por lo que el programa optimizado quedaría:

```
#include <stdio.h>
main() {
     int X,resp;

     scanf("%d",&X);
     if (X>0)
          resp=X*X
     else
          resp=X*(-1);
     printf("%d",resp);
}
```

En general, el uso del condicional **`if`** debe analizarse con cuidado, pues conviene no crear programas muy largos o mal estructurados que dificultan la lectura del mismo.

2.- Hay casos donde es necesario discriminar por un conjunto pequeño de valores discretos que no se encuentran en un rango continuo como en el ejemplo anterior (`X>0` y `X<0`). En esos casos

generalmente se pide realizar cálculos diferentes para cada uno de estos valores. En este tipo de problemas es más elegante usar la instrucción **switch**.

En el siguiente programa se calcula el valor de X^2 si la opción seleccionada es A, $|X|$ si la opción es B y X^3 si la opción es C. Esto se realiza con una instrucción switch. Note el uso de la instrucción **getchar** la cual permite la lectura de un caracter de la entrada estándar.

```
#include <stdio.h>

main() {
     char Opc;
     int X,resp;

     printf("Opciones: ");
     printf("(A) X^2 ");
     printf("(B) |X|");
     printf("(C) X^3 ");
     printf("Introduzca la opción:");
     getchar(Opc);
     printf("Introduzca X:");
     scanf("%d",&X);
     switch (Opc) {
          case 'A': resp=X*X;
               break;
          case 'B': resp=X*(-1);
               break;
          case 'C': resp= X*X*X
               break;
     default:
          printf("Opción errónea");
          break;
     }
     printf("Respuesta = %d",resp);
}
```

Note que con printf también se pueden escribir mensajes. El programa anterior puede ser optimizado a través del caracter "\n", el cual le indica al compilador que debe bajarse de línea. Por lo que las cinco instrucciones para presentar el menú de opciones pueden resumirse en una así:

```
printf("Opciones:\n(A) X^2
                \n(B) |X|
                \n(C) X^3
                \nIntroduzca la opción:");
```

El programa entonces quedaría:

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

```
main() {
   char Opc;
   int X,resp;

   printf("Opciones:\n(A)X²
                  \n(B)|X|
                  \n(C)X³
                  \nIntroduzca la opción:");
   getchar(Opc);
   printf("Introduzca X:");
   scanf("%d",&X);
   switch (Opc) {
     case 'A': resp=X*X;
     break;
     case 'B': resp=X*(-1);
     break;
     case 'C': resp= X*X*X
     break;
   default:
     printf("Opción errónea");
     break;
   }
   printf("Respuesta = %d",resp);
}
```

3.- Dado el siguiente programa ejemplo, el cual utiliza el ciclo **do-while**:

```
#include <stdio.h>
#define N 2
main() {
     int x,y;

     scanf("%d%d",&x,&y);
     do {
          if (x<=y)
               x*=(-1)
          else
               x=N*y;
     } while (x!=y);
}
```

El ciclo **do..while** siempre se ejecuta al menos una vez, pues la **condición de parada** `(x!=y)` se chequea después de ejecutar el cuerpo del ciclo. En este ejemplo se puede observar que si se cumple que `x = -y` el ciclo sólo se ejecuta una vez. En los casos que se cumpla que `x>y` se dice que el programa cae en **"ciclo infinito"** es decir se ejecuta indefinidamente. Es conveniente agregar chequeos a los programas que eliminen la posibilidad de un ciclo infinito. Estas consideraciones también deben tomarse en cuenta al usar los ciclos **while**.

Observe la instrucción "**x*=(-1);**", la cual es una simplificación de la instrucción **"x=x*(-1);".** En C, todas las asignaciones de la forma **"<var> = <var> op <exp>;",** pueden ser simplificadas en **"<var> op= <exp>;".**

4.- El ciclo **for** permite repetir bloques de instrucciones de una manera similar a un `while`, pero dentro de su estructura se puede incluir la inicialización de la variables del `for` y el autoincremento. El siguiente fragmento de programa calcula el producto de los primeros 100 enteros:

```
prod=1;
for (i=1; i<=100; i++)
     prod*=i;
```

Este for se interpreta como el siguiente **while**:

```
i=1;
while (i<=100) {
     prod*=i;
     i++;
}
```

Por lo cual, en el **encabezado del for** se pueden detectar tres áreas principales: una primera área de inicializaciones, una segunda área con la condición de parada del `for` y una tercera área con instrucciones a ejecutar después del cuerpo del `for`. El operador **++** o de incremento unitario puede ser colocado antes o después de una variable. Si se coloca antes de la variable, se dice que es preincremental y su efecto es incrementar en uno la variable y el valor utilizado en la expresión donde resida es el nuevo valor. En el caso de que se coloque después de la variable, se dice que es postincremental; el valor actual de la variable se utiliza en la expresión donde resida y luego se incrementa en uno la variable. Por ejemplo,

```
a=3;
printf("%d", ++a);
/* Escribe 4 en pantalla y el valor de a queda en 4/

a=3;
printf("%d", a++);
/* Escribe 3 en pantalla y el valor de a queda en 4 */
```

Un efecto similar a **++** tiene el operador de decremento unitario **--,** puede ser usado como **predecremental** o **postdecremental.**

Ejercicios:

1.- Qué se escribe en pantalla cuando se ejecutan los siguientes enunciados:

a) `printf("%d",x);`
b) `printf("%d",x+x);`
c) `printf("x=");`
d) `printf("x=%d",x);`

e) `printf("%d=%d",x+y,y+x);`
f) `z=x+y;`
g) `scanf("%d%d",&x,&y);`
h) `/* printf("x+y=%d",x+y); */`
i) `printf("\n");`
j) `printf("*\n**\n***\n****\n");`

2.- Dado el siguiente programa que calcula la suma de una secuencia de enteros hasta encontrar el cero, realice uno equivalente pero utilizando un `do-while`.

```
main() {
     int num, suma;

     suma=0;
     scanf("%d\n",&num);
     while (num<>0) {
          suma += num;
          scanf("%d",&num);
     }
     printf("suma=%d",suma);
}
```

3.- Identifique y corrija los errores en cada una de las siguientes instrucciones de C:
a) `scanf("d",value);`
b) `printf("El producto de %d y %d es"\n,x,y);`
c) `primnom+secnom=sumdenom;`
d) `Scanf("%d",unint);`
e) `Prinft("El valor introducido es:%d\n,&value);`

4.- Escriba un programa en C que solicite al usuario dos números, luego tome dichos números e imprima la suma, el producto, la diferencia, el cociente y el módulo de los números.

5.- Escriba un programa que lea el radio de un círculo y que imprima el diámetro del mismo, su circunferencia y su área. Utilice el valor constante 3,14159 para "pi". Efectúe cada uno de estos cálculos dentro de instrucciones "`printf`" y utilice el especificador de conversión `%f`.

6.- Escriba un programa que imprima sus iniciales en letras de bloque hacia abajo de la página. Por ejemplo, "PT " son las iniciales de Pepe Trueno

```
PPPPPPPPP
    P   P
    P   P
     P P
       T
TTTTTTTTT
       T
```

7.- Dado el siguiente programa que calcula la suma de los primeros N naturales, escriba programas equivalentes con:

a) `do-while`
b) `while`
c) `for` y el operador `++`

```
#define N 5
main() {
    int cont,sum;

    sum=0;
    for (cont=N; cont>=0; cont--)
        sum+=cont;
    printf("suma=%d",suma);
}
```

8.- Observe con cuidado los ejercicios anteriores, y según los resultados de los mismos,

a) Generalice y dé una fórmula para llevar un ciclo `while` a uno `do-while`.
b) Analice cuidadosamente y explique qué diferencias hay con el número de veces que se puede ejecutar un `while` y un `do-while`.

9.- Dado el siguiente programa, indique valores para `X` e `Y` tal que:

a) El ciclo sea ejecutado al menos una vez
b) El programa termine
c) El programa caiga en ciclo infinito.

```
#define M 3
main() {
   float x,y;

   scanf("%f %f",&x,&y);
   while (x!=y) {
          if (x<y)
               x*=(-1)
          else
               x=y+M;
        }
}
```

10.- Realice ejemplos de programas utilizando ciclos donde:

a) Un ciclo nunca sea ejecutado.
b) La ejecución del ciclo sea infinita.

11.- Determine los valores de cada variable, después de que se haya ejecutado el cálculo. Suponga que cuando empieza cada instrucción, todas las variables tienen el valor 5.

a) `product *= x++;`
b) `result = ++x + x;`

12.- Determine la salida para cada uno de los siguientes trozos de código, cuando **x** es 9 **y** es 11, cuando **x** es 11 **y** es 9. Advierta que el compilador de C siempre ignora la sangría y cada el se corresponde al último `if` a menos que se indique lo contrario a través de las llaves `{}`.

```
a) if (x <10)
   if (y > 10)
   printf("*****\n");
   else
   printf("#####\n");
   printf("$$$$$\n");
b) if (x <10) {
   if (y > 10)
   printf("*****\n");
   }
   else {
   printf("#####\n");
   printf("$$$$$\n");
   }
```

13.- Escriba un programa que imprima 100 asteriscos, uno por uno. Después de cada décimo asterisco, su programa deberá imprimir un caracter de nueva línea. (*Sugerencia*: cuente de 1 a 100. Utilice un operador de módulo para reconocer cada vez que el contador llegue a un múltiplo de 10).

14.- Realice un programa en C, que dados dos números enteros `m` y `n`, calcule `m/n` si `m=10`, `m*n` si `m=5`, `m+n` si `m=3`, m^n si `m=2` y en cualquier otro caso imprima los valores de `m` y `n`. Para ello:

a) Utilice la instrucción `switch`.
b) Construya un programa equivalente con instrucciones `if`.
c) Analice ambos programas y diga cuál es más legible. ¿Qué conclusión obtiene?

15.- Señale la salida de cada uno de los siguientes programas ante la entrada que se presenta a continuación. Realice en cada caso la "corrida en frío" y señale los cambios que se podrán introducir en cada uno para mejorarlos.

```
00001.00 00001.00
00008.00 00002.00
00009.00 00003.00
00000.00 12345.67
00666.66 00656.66
```

a)

```
main(){
     float X,Y,Suma,L,K,A,T0,TN;

     printf("%f %f",&X,&Y);
     A=X*Y;
```

```
        L=1;
        Suma=0;
        while (L>0) {
             scanf("%f %f",&L,&K);
             if (L>O)
                  Suma+= L/K/A
             else
                  Suma =Suma
        }
        scanf("%f %f",&T0,&TN);
        printf("El calor transmitido es: %f",(T0-TN)/Suma);
   }
```

b)

```
   main() {
        float X,Y,Suma,L,K,A,T0,TN,Q;

        printf("%f %f",&X,&Y);
        A=X*Y;
        L=3.14159/2.718281-1/3;
        Suma =3*4-12;
        while (L>0) {
             scanf("%f %f",&L,&K);
             Suma=Suma+L/K/A
        }
        scanf("%f %f",&T0,&TN);
        Q=(T0-TN)/Suma;
        printf("El calor transmitido es: %f",Q);
   }
```

16.- ¿Qué es lo que no está bien en la siguiente instrucción?. Vuelva a escribir la instrucción, para que ejecute lo que probablemente el programador intentaba hacer.

```
printf("%d",++(x+y));
```

17.- Escriba un programa que lea tres valores enteros distintos de cero, y que determine e imprima si pueden ser los lados de un triángulo rectángulo.

18.- Dado el siguiente programa, elimine los errores de compilación que aparecen en dicho programa. Use como ayuda un compilador de C.

```
   main(); {
   #define const1 10;
     int var1,var2;
     real var3,var4;
     char var5, var6;

     const1=var2;
     var3=30.0;
```

```
  var5='2';
  var6='L';
  for var4=1; var4<=5;var4++; {
    var2= var2 +2.0;
    printf("El valor de var2 es: %c",var2);
  }
  if (var3>var1) THEN {
    WHILE (var6!=P) {
      printf("El valor de var6 es %c:var6");
      var6+='1';
    }
  };
  else {
    printf("El valor de var 6 es %c",var6);
    var7 =var3+var2;
    var1=var5/var2;
  }
}
```

19.- Despliegue un patrón cuadriculado utilizando seis instrucciones `printf` y a continuación haga el mismo despliegue con el mínimo posible de enunciados `printf`

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

Capítulo 4

Programas Sencillos

Ejemplos:

En el siguiente ejemplo, pueden observarse las diferentes secciones, señaladas en comentarios (`/* */`), de un programa en C que calcula la suma de los n primeros naturales:

```
#include <stdio.h>  /* Uso de librerías externas */
main() {            /* Encabezado del programa principal */
                    /* Declaraciones de Variables */
     int n;              /*n:número a leer*/
     int y;              /*i:contador*/
     int suma;           /*suma: sumatoria de 1 a n*/
                    /* Cuerpo del programa principal /
     printf("Cálculo de la suma de 0 hasta n\n");
     printf("Introduzca el valor de n:");
     scanf("%d",&n);
     i=0;           /*Inicializaciones de variables*/
     suma=0;
     while (i<=n){  /*Ciclo que calcula la sumatoria*/
          suma+=i;  /*Agrego i a la suma que llevaba*/
          i++;      /*Incremento del contador*/
     }
     printf("La suma es: %d",suma");/*Salida del programa*/
}                   /*Fin del programa principal*/
```

Ejercicios:

1.- Escribir programas en C para cada una de las siguientes proposiciones:

a) Convertir una temperatura en grados Fahrenheit a grados Celsius. Utilizar la fórmula:
   `Celsius=(5/9)*(Fahrenheit-32)`
b) Dado el peso de una persona en libras, calcule su peso en kilogramos. NOTA: 1 libra es aproximadamente igual a `0,453592` Kg.
c) Dado el valor de n, calcular la suma de la series: `(1/1)+(1/2)+(1/3)+...+(1/n)` y $(1^1)+(2^2)+(3^3)+...(n^n)$
d) Lea un caracter y deduzca si es vocal, consonante, dígito o caracter especial

2.- Crear un programa que calcule los valores máximos, mínimos y media de tres números leídos en el programa principal.

3.- Escribir un programa que lea N enteros de la entrada y calcule: la suma de los números pares, la suma de los números impares, el promedio de los números pares y el promedio de los números impares.

4.- Elabore un programa que lea pares de números de la entrada hasta que aparezca un cero (0), y que por cada par escriba el resultado de la potenciación con base el primer elemento y exponente el segundo elemento.

5.- Hacer un programa para verificar si una secuencia de números provisto por el teclado está ordenada en forma creciente, decreciente o no. Se deben producir los mensajes correspondientes por pantallas.

6.- Diseñe e implemente un programa para validar la fecha en el siguiente formato: DD/MM/AA. Debe tomar en cuenta los días que tiene cada mes, y los años bisiestos.

7.- Dada una secuencia de números reales, elabore un programa que halle el valor de $f(x)=e^x$ para cada uno de los números reales.

8.- Dados dos pares de enteros elabore un programa que halle el máximo común divisor y el mínimo común múltiplo de ellos.

9.- Escribir un programa que calcule el número de veces que cada letra mayúscula `A..Z` se produce en una línea de texto dada del teclado.

10.- Escriba un programa que realice la conversión de un número binario a un número decimal. Dado un número binario, su equivalente en decimal se obtiene multiplicando cada uno de los bits por la potencia de 2 correspondiente a su posición y sumando los resultados de cada producto. El número binario se lee de la entrada estándar uno a uno los dígitos desde el menos significativo hasta el más significativo. Para demarcar el fin de entreda se utilizará el dígito 9. La salida es el número decimal. Ejemplo de entrada: `0 1 1 0 0 1 9`. Salida: `22`.

11.- Una variante del programa anterior es tener más de un número binario en la entrada, de manera de irlos convirtiendo uno a uno e ir acumulando los resultado para dar un total. Los números binarios estarían separados por el dígito 7 y el fin de la entrada vendrá dado por el número 9.

12.- Escriba un programa que realice la conversión de un número decimal a un número binario.

13.- Dados tres valores `a,b,c` elabore un programa que intercambie los valores de manera que cumplan $a \leq b \leq c$.

14.- Los números perfectos son aquellos enteros que son iguales a la suma de sus divisores sin incluir a él mismo. Ejemplo: `6` es un número perfecto. Elabore un algoritmo que calcule todos los números perfectos menores que un N dado. Nota: se debe realizar el algoritmo con `for`,`do-while` y `while`, y ver la conveniencia de cada uno.

15.- Se tiene una entrada de dígitos entre 0 y 9, precedida por el número total de dígitos a leer. Dada la entrada, se debe calcular el total de número pares, el total de números impares, el promedio de todos los números, la sumatoria de los números pares, la sumatoria de los cuadrados de los números pares y la sumatoria de los números impares.

16.- Simular el siguiente código usando `while` y un `for`

```
a=Formula;
do{
...
a=NuevaFormula;
}while a!=VALOR;
```

17.- Simular el siguiente código usando `do-while` y un `for`

```
a=Formula;
while (a!=VALOR) {
...
a=NuevaFormula;
};
```

18.-La función `ln x` (logaritmo neperiano) puede aproximarse por Series de Taylor como sigue:

$$\ln(x+1) = \sum_{k=1}^{n}((-1)^{k-1}x^k / k)$$

donde **`X`** es un número real entre `-1` y `1`, y **`n`** es un valor entero que indica el número de términos a considerar en Serie. Elabore un programa en C que tome como entrada el valor de `X` y de **`n`**, y produzca como salida la aproximación para la función **`ln`** aplicada a **`X`.** Utilice para ello un ciclo `for`. Escriba la ventajas de usar este ciclo respecto a los otros (`while` y `do-while`).

19.- Modifique el programa anterior para que la serie continúe calculandose hasta que el error con respecto a la función predefinida `log(X)`, tenga dos dígitos significativos. Es decir

`|log(X)-aproximación| < 0,01`

Utilice un ciclo `do-while`. Diga la ventaja de usar este ciclo respecto a los otros.

Capítulo 5

Constructores de Tipos: arreglos y estructuras

Ejemplos:

1.- Además de int, char y float, otros tipos de datos simples son las enumeraciones o tipos **enumerados**, donde se listan las constantes del tipo. Este tipo se declara con la palabra reservada **enum**, como un conjunto de constantes enteras representadas por identificadores. Por ejemplo:

```
enum ComunidadUniversitaria {obrero,administrativo,estudiante,
                             profesor};
```

Los valores de una enumeración se inician con 0 a menos que se inicialice en otro valor, y se incrementan en 1. En el ejemplo anterior, los identificadores son definidos automáticamente en los enteros 0 a 3. Otro ejemplo, donde los identificadores tienen valores del 1 al 6 es:

```
enum Deportes
{Futbol=1,Beisbol,Tenis,Basketball,Natación,SoftBall};
```

Los identificadores en una enumeración deben ser únicos. Los nombres de las constantes no pueden ser modificados en el programa.

2.- Además de los tipos simples, C provee los tipos estructurados los cuales permiten construir estructuras de datos más complejas. Entre ellos están los **arreglos**. Los arreglos ocupan espacio en memoria. En la declaración se especifica el tipo de cada elemento y el número de elementos requerido por cada arreglo. Para declarar un arreglo se escribe lo siguiente

<nombre_de_tipo> <identificador>[<tamaño>];

Un ejemplo de esta declaración sería: `int s[6];`

Con los arreglos se pueden construir y manipular colecciones de datos del mismo tipo como si formaran un solo objeto. Una analogía común con los arreglos son los vectores, los cuales son una colección de elementos del mismo tipo (por ejemplo Reales) que tiene un tamaño y forman un solo objeto el vector. La declaración del tipo vector sería

```
typedef float Vector[100];
```

En esta declaración se puede observar que el vector está formado por 100 elementos todos reales. La declaración de la variable sería

```
Vector v1;
```

El uso de un arreglo es como el de cualquier variable, pero debemos indicar cuál es el elemento que queremos accesar. Esto se realiza a través del **índice del arreglo**, el cual indica la

posición del elemento a usar. Las posiciones en C, siempre empiezan en cero (0) y terminan en la posición *<tamaño> – 1*. En el caso de los vectores, para obtener el valor de la posición 20 y colocarlo en una variable real, se puede hacer en algunas formas equivalentes como

```
y=v1[19];      i=19;          i=1;
               y=v1[i];       y=v1[20-i];
```

Note que en el índice del arreglo (señalado entre corchetes) podemos colocar una constante, una variable o una expresión, siempre que sean de tipo entero que el subrango indicado en la definición del arreglo.

Algunos cuidados que debemos tener con los arreglos son:

- el índice no debe tomar valores fuera del subrango, por lo cual al usar variables o expresiones dentro del índice se debe hacer con precaución;
- el tamaño de los arreglos está limitado por la memoria disponible.

3.- Una de propiedades de los tipos estructurados en C es que éstos pueden ser definidos en función de otros tipos estructurados. Es así como, el tipo de los elementos de un arreglo podría ser un arreglo. En C se simplifican estas construcciones en lo que se denominan **arreglos bidimensionales** los cuales son análogos con las matrices.

La definición de una matriz quedaría

```
float Matriz [100][150];
```

En este caso se dice que la matriz (arreglo bidimensional) es de tamaño 100x150 y sus elementos son números reales.

Después de la declaración de la variable `m` como Matriz se puede usar dentro de expresiones a través de dos índices, uno por cada dimensión del arreglo, como por ejemplo

```
m[10][35] = m[10][36]+m[11][35];
```

4.- Otro tipo estructurado provisto por C son las estructuras o **`struct`**. Estos permiten manipular elementos de tipos diferentes como si formaran un solo objeto. A estos elementos se les denomina **campos** de la estructura. Las estructuras se definen usando la siguiente instrucción:

```
struct <nombre_del_tipo> {
        <tipo> <nombre_campo>;
                    ...
        <tipo> <nombre_campo>;
} <lista de identificadores>;
```

Para usar una estructura debe especificarse todo el camino donde se localiza el campo. Por ejemplo, dada la definición de un tipo estructura que almacene una fecha

```
typedef struct Fecha {
     int día;
```

```
        enum mes {ene,feb,mar,abr,may,jun,
                  jul,ago,sep,oct,nov,dic};
        int año;
    }Tipo_fecha;
```

Se puede asignar una variable con la fecha de cumpleaños de la siguiente manera:

```
Tipo_Fecha cumpleaños;
     ...
cumpleaños.día = 06;
cumpleaños.mes = jul;
cumpleaños.año = 1996;
```

Como se mencionó anteriormente, todo tipo estructurado puede estar definido en función de otros tipos estructurados. Por ello, se pueden construir estructuras de estructuras, arreglos de estructuras, estructuras de arreglos y así sucesivamente.

Ejercicios:

1.- Usando tipos enumerados, arreglos y estructuras, dé definiciones de tipos para:

a) Colores primarios
b) Días de la semana
c) Deportes acuáticos
d) Vocales y consonantes
e) Cargos de profesores: instructor, asistente, asociado y titular
f) Meses del año
g) Trimestre Ene-Abr 94
h) Mayúsculas y minúsculas
i) Minoría de Edad

2.- Implemente un programa que invierta un número entero que tenga a lo sumo 10 cifras.

3.- Realice un programa en C que pida al usuario un día de la semana y despliegue el plato principal del almuerzo del comedor de la Casa del Estudiante. Utilice tipos enumerados.

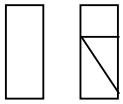
4.- Diseñe e implemente un programa para intercalar dos secuencias de números enteros S_1 y S_2 que dé como resultado una secuencia S ordenada ascendentemente sin elementos repetidos. Inicialmente la secuencia S_1 tiene N_1 elementos y la secuencia S_2 tiene N_2 elementos y ambas están ordenadas.

5.- Escribir un programa que calcule unión, intersección y diferencia de dos conjuntos de enteros A y B de tamaño máximo 10. El programa debe permitir la inicialización de los conjuntos desde el teclado. Ejemplo de corrida:

```
Introducir número de elementos de A: 6
Introducir los elementos de A: 0 1 50 3 21 18
Introducir número de elementos de B: 8
Introducir los elementos de B: 6 7 50 4 18 21 1 20
La unión de A con B es: 0 1 50 3 21 18 6 7 4 20
```

```
La diferencia A menos B es: 0 3
La intersección de A y B es: 1 50 18 21
```

6.- Implemente un programa en C que lea dos puntos en el espacio (pares ordenados) que definen un rectángulo y almacene en una estructura de datos los puntos que definen el cuadrado y los dos rectángulos que encajan perfectamente en el rectángulo dado, tal como se muestra en la siguiente figura:



7.- Un encriptador oculta mensajes de tamaño N en un arreglo de tamaño 3*N de caracteres siguiendo el programa:

- En las N primeras casillas se almacenará una permutación al azar del mensaje original.
- Las 2*N casillas restantes se interpretarán como una serie de pares desplazamiento-dirección, donde desplazamiento indica el número de caracteres a avanzar ('A':1 , 'B':2 ,..) y dirección indica hacia donde va el desplazamiento ('F':adelante, 'B':atrás).

De modo que para decodificar el mensaje, se partirá desde la casilla N y se seguirá la traza indicada por los pares desplazamiento-dirección hasta el último de estos.

Ejemplo:

r	o	c	r	a	B	B	B	F	A	B	C	B	A	F
1	2	3	4	5	6	7	9	9	10	11	12	13	14	15

es una codificación del mensaje "carro" con N=5. Escriba una función DECODIFICADOR que reciba como parámetro un arreglo de tamaño 3*N y escriba en pantalla la secuencia decodificada.

8.- Se representan polinomios de grado N en arreglos de tamaño N+1 de la siguiente forma:

$$p(x) = C_n*X^n + C_{n-1}*X^{n-1} + \ldots + C_0$$

Escribir un programa que evalúe un polinomio en dicha representación en un punto "a":

a) Calculando la suma $C_n*X^n + C_{n-1}*X^{n-1} + \ldots + C_0$
b) Usando la forma factorizada de Horner:

Iterar desde 0 hasta N: I_0: $P = C_n$

I_k: $P = a*P + C_{n-k}$

9.- Realice un programa que lea dos vectores de la entrada y produzca como salida el producto escalar y el producto vectorial de los mismos.

10.- Realice una definición para un `BYTE`: secuencia de ocho bits, un `BIT` puede ser `0` ó `1`.

11.- Utilizando la definición anterior, realice un programa que permita:

a) La introducción de bytes.
b) Desplazamiento hacia la izquierda y hacia la derecha de los bits de un byte.
c) Rotación hacia la izquierda y hacia la derecha de los bits de un byte.

12.- Se tienen N temperaturas se desea calcular su media, la desviación estándar y cuáles de las temperaturas son inferiores a la media.

13.- Elabore un programa que inserte un elemento en un arreglo ordenado. Pensar en un método distinto al secuencial.

14.- Realizar un programa que permita calcular el producto escalar de dos vectores de dimensión 2000 en los cuales al menos 99% de sus elementos son cero.

15.- Se desea realizar un programa donde se tenga como entrada un número binario y se obtenga su representación en decimal, octal, hexadecimal. La longitud del número binario es de ocho dígitos.

16.- Realice un programa que elimine elementos duplicados de una lista y por cada elemento indique la cantidad de veces que se repite en la lista.

17.- En una tabla de tamaño MxN se almacenan m palabras de n caracteres. Escriba una función que lea una palabra, y ubique su posición dentro de la tabla. De no existir, buscar la posición de la palabra más parecida a la original.

18.- Escribir un programa que transponga una matriz sin emplear una matriz auxiliar.

19.- Escribir un programa que copie un arreglo A de dimensión N1 = N*N en una matriz de M orden NxN, asumiendo que los datos de A son la concatenación de las filas de M.

20.- Escribir un programa que muestre en pantalla los elementos de una matriz al recorrer esta en espiral partiendo desde el elemento 1,1 en sentido horario.

21.- Escribir un programa que verifique que una matriz M de orden N*N es una matriz de permutación. (Una matriz de permutación es una matriz resultante de aplicar una operación básica de fila (intercambio, multiplicación o suma) sobre la matriz identidad).

22.- Se desea que usted almacene M llamadas telefónicas. De cada llamada se almacenan los siguientes datos: la extensión, el número marcado, la duración, la fecha y el destino; además por razones operaciones se quiere que la fecha contenga: hora, minutos, segundos, día, mes y año en que fue realizada. Sugerencia: realice una definición de tipos que facilite el almacenamiento y acceso de los datos.

23.- Usando la definición del ejercicio anterior, calcule el número de llamadas realizadas en el mes de diciembre.

24.- En un colegio la información de la comunidad es guardada en un arreglo, llamado `COLEGIO`, cuyos componentes son del tipo `PERSONAL`. Los datos del personal son: nombre, apellido, edad, dirección y otros datos que dependen de si la persona es profesor o alumno. En el caso de un profesor se almacena la categoría, el salario y el turno; para los alumnos se almacena el curso, la sección y la nota promedio.

a) De acuerdo a lo explicado anteriormente, defina los tipos necesarios.
b) Usando la parte a, calcule el número de alumnos y profesores del `COLEGIO`.
c) Liste el nombre, apellido y promedio de todos los alumnos de la sección dos de cuarto año.

25.- Defina el tipo de datos `FichaComUniv`, el cual puede almacenar los datos de cualquiera de los miembros de la Comunidad universitaria. Existen cuatro clases de miembros universitarios: Estudiantes, Profesores, Obreros y Empleados. Para todos se almacenan datos personales como: cédula, combres, apellidos, edad, teléfono, dirección, sexo, número de hijos. Pero hay algunos datos que son propios a cada clase, entre ellos:

- Para los estudiantes: carnet, carrera, número de créditos aprobados, número de créditos inscritos, número de créditos reprobados, número de trimestres inscritos, índice académico
- Para los profesores: departamento académico al que pertenece, fecha de ingreso a la universidad, cargo desempeñado (instructor, asistente, agregado, asociado, titular), dedicación (convencional, integral o exclusiva), sueldo.
- Para los obreros: empresa contratista a la que pertenece, sueldo, categoría (I,II,III,IV,V,VI)
- Para los empleados: departamento administrativo al que pertenece, fecha de ingreso, sueldo, cargo.

Realice un programa que permita llenar los datos de una ficha. Use menues para seleccionar tipos enumerados.

26.- Escriba un programa que permita realizar el producto escalar de dos vectores de dimensión `20000` en los cuáles, al menos 99% de sus posiciones son ceros.

27.- Proponga estructuras de datos eficientes que simulen las siguientes situaciones:

a) Un cuadro del 5 y 6
b) Información académica de un estudiante
c) La memoria de un computador
d) Una matriz en la que se desea saber si es invertible, si es identidad o normal, su determinante, si es simétrica
e) Un ecuación de grado `n`

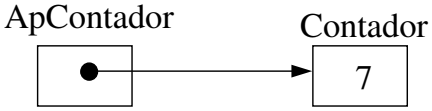
28.- El profesor Gómez desea procesar información sobre los veinte exámenes aplicados a su curso de Semiótica. Cada examen corresponde a un estudiante, el cual es identificado por su número de carnet. El profesor Gómez redondea la calificación del examen, pero quiere mantener también la nota real antes de redondear con sus cifras decimales. La escala de calificación para estos exámenes es sobre 20. También le interesa saber el número de items del examen que cada estudiante contestó y de ellos cuántos fueron acertados. Los estudiantes se categorizan según su plan de estudios en estudiantes de pregrado y postgrado, y según su situación en la universidad como estudiante regular, ocasional y por equivalencia. Se desea que diseñe la estructura de datos necesaria para modelar esta información.

Capítulo 6

Apuntadores y Arreglos

Ejemplos:

1.- Los apuntadores son variables que contienen direcciones de memoria como sus valores. Por lo regular una variable contiene directamente un valor específico. Un apuntador contiene la dirección de una variable que contiene un valor específico. Ejemplo:



`Contador` se refiere directamente a la variable cuyo valor es 7, mientras que **`ApContador`** se refiere en forma indirecta a la variable cuyo valor es 7.

Los apuntadores deben ser declarados, como toda variable, antes de ser usados. Para ello se coloca

<tipo> *<nombre del apuntador>;

donde **<tipo>** puede ser cualquiera de los tipos de datos de C.

2.- Si queremos declarar un apuntador a un entero (`int *`), se haría la siguiente declaración:

```
int *ApContador, Contador;
```

lo cual indica que **`ApContador`** será un apuntador a un objeto de tipo entero. Note que en la misma declaración se indica que **`Contador`** es un entero, lo cual señala que para que se declare un apuntador debe colocarse antes del nombre de la variable un asterisco **(*)**.

Para inicializar un apuntador se puede usar la constante `NULL` definida en el archivo <stdio.h> , lo cual indica que el apuntador tiene valor cero o apunta a nada.

3.- El operador unario **`&`**, conocido como operador de dirección, regresa la dirección del operando. Por ejemplo

```
ApContador = &Contador;
```

asigna la dirección de la variable **`Contador`** al apuntador **`ApContador`**, y se dice que **`ApContador`** apunta a **`Contador`**. La representación gráfica es mostrada en la figura anterior.

4.- Una representación del apuntador en memoria, suponiendo que la variable entera `Contador` se almacena en la posición `612897`, y la variable apuntador se almacena en la `234500` sería

El operador de dirección `&` solo puede aplicarse a variables, no pueden ser operandos de este operador: constantes, expresiones o variables de tipo archivo.

Note que cuando hacemos una lectura con la función predefinida scanf, se coloca en las variables donde queremos que quede la lectura un operador de dirección ya que queremos que el valor de la variable salga modificado con el valor de la lectura. Ejemplo:

```
scanf("%d",&Contador);
```

en `Contador` queda almacenado el valor entero leído, en caso de no colocar el operador `&` antes de la lectura la variable `Contador` no sería modificada.

5.- El operador unario ***,** conocido como operador de indirección, regresa el valor del objeto hacia el cual el operando apunta. Por ejemplo **`printf("%d",*ApContador);`** imprime el valor de la variable `Contador`, es decir, `7`.

6.- Se pueden crear estructuras de datos dinámicamente usando la instrucción `malloc`. El límite de asignación dinámica de la memoria puede ser tan grande como la totalidad de memoria física disponible en la computadora. A menudo, los límites son muchos menores, porque la memoria disponible debe ser compartida entre muchos usuarios.

La función **`malloc`** toma como argumento el número de bytes a asignarse y regresa un apuntador del tipo `void*` (apuntador a `void`) a la memoria asignada. Un apuntador `void*` puede asignarse a una variable de cual tipo de apuntador. Normalmente la función malloc se utiliza conjuntamente con el operador **sizeof**, el cual devuelve el tamaño en bytes de una estructura dada. Por ejemplo: **`ApNuevo = malloc(sizeof (struct Nodo));`** evalúa **`sizeof(struct Nodo)`** para determinar el tamaño en bytes de una estructura del tipo **`struct Nodo`**, y luego asigna al apuntador `ApNuevo` la dirección de un bloque de memoria del tamaño dado por `sizeof`.

La función **`free`** cancela la asignación de memoria, regresándola al sistema. Para liberar memoria asignada dinámicamente mediante la llamada a `malloc` previa se usa

```
free (ApNuevo);
```

Ejercicios:

1.- Conteste cada una de las siguientes preguntas. Cada parte del ejercicio deberá utilizar los resultados de las partes anteriores donde sea apropiado:

a) Declare un arreglo de tipo `float`, llamado `numbers` con `10` elementos, e inicialice los elementos a los valores `0.0,1.1,2.2,...9.9`. Suponga que la constante simbólica `SIZE` ha sido definida como `10`.
b) Declare un apuntador `nPtr`, que apunte a un objeto de tipo `float`.
c) Imprima los elementos del arreglo `numbers`, utilizando notación de subíndice de arreglos. Utilice una estructura `for`, y suponga que se ha declarado la variable de control entera `i`.

d) Proporcione dos instrucciones por separado, que asignen la dirección inicial del arreglo `numbers` a la variable de apuntador `nPtr`.
e) Imprima los elementos del arreglo `numbers`, utilizando únicamente la variable `nPtr`.
f) Imprima los elementos del arreglo `numbers`, utilizando el nombre del arreglo como un apuntador.
g) Imprima los elementos del arreglo `numbers` mediante subíndices del apuntador `nPtr`.
h) Suponiendo que `nPtr` apunta al principio del arreglo `numbers`, ¿cuál es la dirección referenciada por `nPtr+8` ? ¿ cuál es el valor almacenado en esa posición?
i) Suponiendo que `nPtr` apunta a `numbers[5]`, ¿qué dirección es referenciada por `nPtr=nPtr-4` ? ¿cuál es el valor almacenado en esta posición?

2.- Encuentre el error en cada uno de los segmentos de programas siguientes. Suponga:

```
int *zPtr;
int *aPtr = NULL;
void *sPtr = NULL;
int number,i;
int z[5] = {1,2,3,4,5};
sPtr=z;
```

```
a) ++z;
b) number = zPtr;      /* obtiene el primer valor del arreglo */
c) number = *zPtr[2]; /* asigna el valor 3 a number*/
d) number = *sPtr;     /* asigna lo apuntado por sPtr a number */
e) for (i=0;i<=5;i++) /* imprime el arreglo z */
     printf("%d",zPtr[i]);
```

3.- Para cada uno de los siguientes enunciados, escriba el trozo de código correspondiente. Suponga que se han declarado las variables **num1** y **num2** de tipo flotante.
a) Declare la variable **fPtr** como apuntador a un flotante
b) Asigne la dirección de la variable **num1** a la variable **fPtr**
c) Imprima el valor del objeto señalado por **fPtr**
d) Asigne el valor del objeto señalado por **fPtr** a la variable **num2**
e) Imprima el valor de **num2**
f) Imprima la dirección de **num1**. Utilice el especificador de conversión `%p`
g) Imprima la dirección almacenada en **fPtr** utilizando `%p` ¿Es el valor impreso el mismo de la dirección de **num1**?

4- Haga la simulación de arreglos sin tamaño fijo usando apuntadores. Para ello pida al usuario un valor N, que indique el tamaño del arreglo. Luego lea los N elementos del arreglo introducidos por el usuario y los almacene en la estructura de apuntadores usando la instrucción `malloc`. A continuación imprima el arreglo en orden inverso liberando el espacio reservado para cada elemento con la función free.

5.- Escriba un programa que lea dos listas ordenadas de enteros positivos. El tamaño de las listas es cualquiera. El usuario determina el final de la lista con un valor negativo. El programa debe producir como resultado una sola lista ordenada de enteros positivos que combine ambas listas.

6.- Construya una estructura de datos que almacene en forma eficiente matrices triangulares inferiores. Realice un programa que lea dos matrices de tamaño 5x5, las almacene en una estructura de datos eficiente y devuelva la multiplicación de dichas matrices.

7.- Qué modificación debe realizar a la estructura de datos del ejercicio anterior para que almacene en forma eficiente matrices diagonales. Cómo puede aprovechar esta estructura para la multiplicación.

8.- Construya una estructura de datos que almacene el nombre y el índice de N estudiantes dados por el usuario y produzca como resultado el índice promedio de dichos estudiantes. N es dado por el usuario antes de introducir los datos.

9.- Realizar un programa en C que resuelva un sistema de N ecuaciones y N incognitas

Capítulo 7

Funciones, Alcance de Variables y Pasaje de Parámetros

Ejemplos:

Las funciones son **subprogramas** que pueden ser definidos dentro de un programa cuya utilidad principal es ayudar a construir programas bien estructurados donde se facilite la lectura y mantenimiento de los mismos.

Todos los subprogramas en C, son funciones, por lo que los procedimientos se definen como funciones que no retornan ningún valor, los cuales se declaran **`void`**.

Las invocaciones a procedimientos pueden usarse como instrucciones simples, que pueden ser utilizadas en diferentes lugares del programa principal. Al ejecutarse la invocación o llamada al procedimiento lo que se ejecuta es el cuerpo del subprograma llamado. En el caso de las funciones, sus invocaciones se realizan dentro de expresiones e igualmente la ejecución de una invocación a función ejecuta el cuerpo de dicho subprograma.

1.- El siguiente programa intercambia 50 pares de valores introducidos por el teclado. Para ello utiliza una **función** sin parámetros llamada Intercambio.

```
#include <stdio.h>

void Intercambio(int *, int *);

main ()
{
     float x,y,temp;
     int i;

     for (i=1; i <= 50; i++)
     {
          printf("Introduzca dos números:");
          scanf("%d %d",x,y);
          intercambio(&x,&y);
          printf("%d %d",x,y);
     }
}
void Intercambio(int *, int *)
{
     int temp;
     temp=*x;
     *x=*y;
     *y=temp;
}
```

La función Intercambio es un procedimiento pues el tipo retornado es **`void`**. Este procedimiento tiene dos **parámetros variables o por referencia**, lo cual se determina por los apuntadores a las variables `x` e `y`. Esto indica que los valores iniciales de `x,y` pueden ser modificados dentro del procedimiento, en efecto lo es, obsérvese el cuerpo del mismo. Es de notar que los procedimientos son invocados como una instrucción simple dentro del programa.

A las variables `x,y` dentro de la invocación al procedimiento `Intercambio` se les denomina **parámetros reales**. Las variables `a,b` en la declaración del procedimiento se denominan **parámetros formales**.

Los parámetros reales que no se pasan usando apuntadores son llamados **parámetros por valor**, se utilizan cuando los valores son únicamente de entrada, por lo que no deben ser modificados dentro del procedimiento, y su valor inicial se mantiene aún después de la llamada al procedimiento.

2.- El siguiente ejemplo muestra la declaración de una **función** que calcula el máximo entre dos números reales, en la cual se puede observar dos parámetros de entrada de tipo `float`. Además se observa que la función tiene un tipo de retorno `float`.

```
float máximo(float x, float y)
{
      if  (x>y)
           maximo=x
      else
           maximo=y;
}
```

Las funciones, a diferencia de los procedimientos, son invocadas dentro de expresiones. Como ejemplo de expresiones tenemos: el lado derecho de una asignación, un parámetro actual en una invocación a subprograma, una condición.

Ejercicios:

Los siguientes ejercicios deben realizarse utilizando procedimientos y funciones, inclusive aquellos que pidan sólo elaborar un programa.

1.- Diseñe e implemente un programa que dada la fecha actual y la fecha de nacimiento de una persona, las valide y determine la edad. Sugerencia: utilice un procedimiento que valide la fecha en formato DD/MM/AA.

2.- Escribir una función que reciba la longitud y anchura de una habitación y retorne la superficie, con sus respectivas cifras decimales.

3.- Escriba una función que reciba como parámetro dos valores M y N, y retorne M^N como resultado. Use solamente multiplicaciones.

4.- Escriba una función que calcule la división entera `X/Y` y devuelva el cociente y el resto de la división. Utilice solamente restas sucesivas.

5.- Escriba una función que dado un arreglo de caracteres retorne el tamaño del mismo.

6.- Escriba una función que dada un arreglo de enteros devuelva el promedio de la lista, el elemento más cercano al promedio y la posición de dicho elemento.

7.- Escriba un trozo de código para cada una de las siguientes proposiciones:

a) El prototipo de una función llamada **`cambio`** que tome como parámetro dos apuntadores **`x,y`** a números de punto flotante.
b) El prototipo de una función llamada **`evaluar`** que regresa un entero y toma como parámetros el entero **`x`** y una función llamada **`poly`**. La función **`poly`** toma un parámetro entero y regresa un entero.
c) El prototipo de una función llamada **`cero`**, que toma como parámetro un arreglo de enteros **`long`** llamado **`MaxEnteros`**.

8.- Escribir un procedimiento que lea un caracter y deduzca si es vocal, consonante, dígito o caracter especial.

9.- Crear un programa que calcule los valores máximos, mínimos y media de tres números leídos en el programa principal mediante las funciones Máximo,Mínimo y Media.

10.- Escriba un programa que, dada una secuencia de n enteros, calcule

a) La cantidad de números positivos
b) La cantidad de números negativos
c) Los nulos que hay en la secuencia.

11.- A continuación se presentan varios programas ilustrativos de lo que ocurre al combinar las reglas de alcance de variables y pasaje de parámetros en una forma no muy organizada. Su actividad concreta es dar los resultados de la corrida de estos programas atendiendo a dichas reglas.

a)

```
void pr(int x,int y,int *z);

main() {
   int a=5,b=8,c=3;
   pr(a,b,&c);
   pr(7,a+b+c,&a);
   pr(a*b,a /b,&c);
}

void pr(int x,int y,int *z)
{
     *z=x+y+*z;
```

```
     write(x,y,*z);
}
```

b)

```
void r1(int *a; int b);
int r2(int i,int a,int k);
void r3(int *a, int i, int k);

main()
{
   int k=1,x=2;

   r1(&x,x,k);
   printf("%d",x);
}

void r1(int *a, int b,int y)
{
   int i=0;

   a=2*b;
   r3(&i,a,y);
   printf("%d %d",x,y);
}

int r2(int i,int a,int k)
{
   i+=3;
   if (k <= 2)
      printf("%d %d",i,k)
   return(i+k+a);
}

void r3(int *a, int i, int k)
{
   int x,y;
   *a=i+k;
   x=r2(*a);
   y=r2(2*(*a));
   printf("%d %d",x,y);
}
```

c)

```
void p1(int *i,int j,int k);
void p2(int h,int *j,int k);
void p3(int *i);
```

```
main()
{
   int i,j,k;
   i=0;
   j=1;
   k=2;
   p2(0,&k);
   p2(1,&i);
   p2(2,&j);
}

void p1(int *i,int j,int k)
{
   (*i)++;
   printf("%d %d %d",*i,j,k);
}

void p2(int h,int *j,int k)
{
   int i;
   i=j;
   if (h==0)
      p1(*j,j,k)
   else
      if (h==1)
     p1(&i,*j,k);
      else
          p3(&i,*j,k);
   printf("%d %d %d",i,j,k);
}

void p3(int *i)
{
  (*i)++;
}
```

12.- ¿Es este un programa correcto en C?, de no serlo señale los errores e indique posibles correcciones. Si fuera correcto, entonces haga la corrida del programa

```
main ()
{
   int f=2,h=10;
   a(f,h,'1');
   a(h,f,'2');
}
```

```
void D int p
{
   f=f+1;
   p=p+f;
   printf(f);
}

void int A(int *x,int *y,char s)
{
   int b,c;

   b=x;
   if (G(b) >= 40)
      c=D(-5);
   else
      c=D(b*2);
}

void G(int *r)
{
   int a,f;
   a=10;
   f=a*a;
   r=r*a;
   printf(r);
   return(r);
}
```

13.- Escriba un programa que calcule los n primeros números de Fibonacci. Los números de Fibonacci cumplen con: `F(0)= F(1)= 0 y F(n)= F(n-1) + F(n-2)  n≥2.`

14.- Escriba una función que calcule el factorial de un número. El factorial se define como:

`n! = n*(n-1)*(n-2)*..*1  n! = n*(n-1)!  n≥1  0! = 1`

15.- Usando la función anterior escriba la función combinatorio de M en N recibidos como parámetros.

16.- Escriba un procedimiento que reciba una cantidad entera N, que representa un monto en bolívares, y lo desglose en billetes y monedas de la siguiente denominación: 1000, 500, 100, 50, 20, 10, 5, 2 y 1.

17.- Dada una sucesión de números comprendidos entre 1 y 100 escribe un programa que calcule todos los divisores de cada número.

18.- Escriba una función que dados los límites de un intervalo y un número real, diga si el número pertenece al intervalo.

19.- Para el siguiente programa, dé la salida del mismo.

```
main() {
   int t=2,c=0, b=0, n=1;
   P(n,t);
}
void Q(int x,y)
{
   b=c;
   printf("%d %d %d",x, y, b);
}
void P(int a, b)
{
   int c;
   n++;
   a++;
   c=b;
   printf("%d %d %d",n, a, b);
   Q(c,b);
}
```

20.- Escriba una función que dados tres pares (x_1,y_1), (x_2,y_2) y (x_3,y_3) los cuales representan los vértices de un triángulo indique si el triángulo es válido y a cuál tipo pertenece: isósceles, escaleno o equilátero.

21.- Dados el tipo de una figura geométrica (círculo o cuadrado) y sus datos correspondientes, implemente una función que calcule el área de la figura.

22.- Dados tres valores `a,b,c` elabore un procedimiento que los intercambie de tal forma que se cumpla que $a \leq b \leq c$.

23.- Escribir un procedimiento que reciba dos variables como parámetros, e intercambie sus valores.

24.- Se desea implementar un programa en C que reciba dos números enteros positivos `N` y `B`, y calcule (de manera iterativa, usando únicamente sumas y restas): el número natural `L`, que es la parte entera del logaritmo de `N` en base `B`, el número `A` como la potencia `L` de `B`. Luego escriba los valores `L`, `A` y `N-A`

25.- Escriba un programa que calcule la función `seno(x)` usando la fórmula $x/1! - x^3/3! + x^5/5! - x^7/7! \ldots$ con el número de términos dados por el usuario. Para ello, use una función que calcule x^y.

26.- Escriba un procedimiento que permita al usuario jugar al ahorcado con una palabra introducida como parámetro.

27.- ¿Cuál es el valor que retorna la siguiente función?. Explique los efectos de borde que se están generando con las siguientes invocaciones: `A=F(A,A);A=F(A,B);A=F(B,C);`

```
int F(int *x,int *y)
{
   y=x;
   x=y;
}
```

28.- Describa en que casos los siguientes fragmentos calculan el mismo valor y valores distintos de C. Explique.

a)
```
A=f1(x);
B=f2(x);
C=A&&B;
```
b)
```
C=f1(x)&&f2(x);
```
c)
```
C=f2(x)&&f1(x);
```
d)
```
A=f2(x);
B=f1(x);
C=A&&B;
```

29.- Verifique el ejercicio anterior, con `x<=1` y con `x>=2`, para el caso en `f1` y `f2` sean:

```
#define BOOLEAN int
#define TRUE 1
#define FALSE 0

BOOLEAN f1(int x)
{
   x++;
   return (TRUE);
}

BOOLEAN f2(int x)
{
   if (x<=2)
      return(TRUE);
   else
      return(FALSE);
}
```

30.- Realice un procedimiento para que, dado un texto encriptado con una clave, y la clave de encriptamiento, produzca el texto original.

Ej: `H M H O S Ñ R`

Función de Encriptamiento: `Letra + clave  Clave=3`

Resultado: E J E M P L O

31.- Escribir un procedimiento INVERTIR que reciba como parámetro un arreglo de enteros de tamaño N, que invierta el contenido de éste sin emplear otro arreglo auxiliar.

32.- Dados a_i, b_i, tales que $p(x) = \Sigma_{0 \le i \le n} a_i x^i$ y $q(x) = \Sigma_{0 \le i \le n} b_i x^i$ con $m, n > 0$ y $n > m$, escriba procedimientos en C para calcular:

a) r(x)= p(x) + q(x)
b) r(x)= p(x) * q(x)
c) p(x) para un x dado.

33.- Escriba un procedimiento para determinar si una secuencia de caracteres es palíndrome, es decir, si es equivalente leerla de izquierda a derecha y de derecha a izquierda. Ejemplos: arepera, radar, 101, ababbaba.

34.- Dado el siguiente programa:

```
main() {
   int i;
   int A[3];

   for (i=0; i< 3; i++) do
      A[i]=0;
   A[1]= 10;
   i=1;
   P(A[i]);
}

void P(int v);
{
   v+=1;
   A[i]= 5;
   i=3;
   v+=1;
}
```

Mostrar los resultados producidos por cada llamada en el arreglo A y en la variable i, suponiendo pasaje de parámetros por valor y luego pasaje de parámetros por referencia.

35.- Realice un procedimiento que dado un arreglo A de (n*(n+1))/2 elementos almacene sus elementos en una matriz de nxn como muestra en el ejemplo:

Si n=4 y el arreglo fuese

```
12 24 15 13 16 9 1 2 0 3
```

La matriz debe quedar así:

```
12   0    0    0
24   15   0    0
```

```
13   16   9    0
1    2    0    3
```

36.- Implemente un procedimiento para sumar dos matrices `nxm`.

37.- Implemente un procedimiento que tenga como entrada el resultado de un juego de "La Vieja" y determine cuál es el ganador (`0` ó `X`)., por ejemplo así:

```
0  0  X
X  0  X
0  X  X
```

el ganador es `X`

38.- Hacer un procedimiento que reciba una matriz nxn de enteros, calcule y escriba la suma de todas las filas, las columnas, las diagonales con pendiente `1` y las diagonales con pendiente `-1` de dicha matriz. Por ejemplo

Entrada:

```
1    5    7
11   10   4
3    2    1
```

Salida:

suma de las filas: `13  25  6`
suma de las columnas: `15 17 12`
suma diag. pend 1: `1 16 20 6 1`
suma diag. pend -1: `3 13 12 9 7`

39.- Escribir un procedimiento `MULTIPLICIDAD` que reciba como parámetros una matriz de `M` orden `NxN` y un arreglo `V` de orden `N` que devuelva el número de veces que `V` aparece como fila de `M`.

40.- Escriba una función `ES_DIAG` que reciba como parámetro una matriz `M` de orden `NxN` que devuelva `TRUE` si `M` es una matriz diagonal.

41.- Realice una función que dada una matriz de tamaño `MxM`, un número que indique una de sus diagonales y un valor entero, devuelva si todos los elementos de la diagonal son iguales a dicho valor. Suponga que la diagonal principal es la número cero, la inferior a esta es -1 y así sucesivamente en orden decreciente; y las diagonales superiores a la principal siguen un orden ascendente a partir de 1. Utilice esta función para hacer un programa que calcule si la matriz es nula y si es identidad.

42.- Se desea mantener la siguiente información acerca de los jugadores que participan en un campeonato de beisbol. Con el fin de llevar estadísticas de dichos jugadores se desean guardar los siguientes datos: nombre y edad del jugador, nombre del equipo al que pertenece, si juega o no en las grandes ligas, si es pitcher(número de juegos ganados en la última temporada, número de juegos perdidos en la última temporada, efectividad) y si no es pitcher (promedio de bateo en la última temporada, número de carreras impulsadas en la última temporada, número de bases robadas en la última temporada, número de cuadrangulares en la última temporada, número de hits conectados en la última temporada).

a) Diseñe una estructura que permita mantener estos datos en forma eficiente (suponga que el número máximo de jugadores es MAXJUG).
b) Codifique un procedimiento que permita rellenar la estructura definida en la parte a).
c) Codifique un procedimiento que permita actualizar los datos de dicha estructura en base a los resultados obtenidos en un juego.

43.- Con las siguientes declaraciones, realice procedimientos que respondan lo siguientes:
a) ¿Cuántos empleados mayores de 50 años están divorciados?
b) ¿Cuál es el porcentaje de mujeres casadas?
c) ¿Cuál es porcentaje de empleados solteros?
d) ¿Cuántas mujeres solteras están entre los 20 y 30 años de edad?

```
#define N 100
typedef enum Estado {Soltero,Casado,Viudo,Divorciado};
typedef struct Fecha {
   int Día;
   int Mes;
   int Año;
} Tipo_Fecha;

typedef struct Empleados {
   char Apellido[20],Nombre[20];
   Fecha FechaNac;
   enum Sexo {Masculino,Femenino};
   char EdoCivil;
}Tipo_Emp;

Empleados Lista_Emp[N];
```

Capítulo 8

Funciones de la Librería Estándar
Archivos

Ejemplos:

1.- Las principales funciones de la librería estándar son **`printf`** (para escritura con formato) y **`scanf`** (para lectura con formato). Ambos utilizan un string con especificadores de conversión que indican el formato, `%s` para string de caracteres, `%d` para enteros, `%c` para caracteres, `%f` para reales, y `%n` almacena el número de caracteres leídos hasta el momento. Algunos ejemplos se presentaron en el capítulo 2 (Instrucciones simples y condicionales).

2.- A través de los **archivos** se pueden almacenar de forma persistente una cantidad cualquiera de elementos de un mismo tipo. Los archivos se pueden manipular a través de una variable conocida como **File Pointer**, la cual señala en todo tiempo el elemento visible actualmente del archivo, ya que sólo es visible un elemento a la vez. La declaración de una variable file pointer de tipo texto sería **`FILE *fp;`**

Para usar un archivo debe abrirse previamente con la instrucción **`fopen`**, la cual devuelve el apuntador a una estructura de tipo `FILE` o file pointer. En esta instrucción debe especificarse el nombre del archivo que se va abrir y el modo de apertura: "**r**" abierto para lectura, "**w**" creado para escritura, "**a**" para agregar al final, "**r+**" abierto para lectura y escritura, "**w+**" creado para lectura y escritura, "**a+**" para lectura o agregar al final. En la siguiente instrucción se abre el archivo cliente.dat para lectura: **`cfPtr = fopen("cliente.dat","r")`**

Con las funciones **`fscanf`** y **`fprintf`** se pueden leer o escribir valores en un archivo, y finalmente se debe cerrar para garantizar que las modificaciones realizadas no se pierdan con la función **`fclose`**.

3.- El siguiente programa crea un archivo de enteros positivos, provistos por el teclado hasta que se introduzca un valor negativo.

```
main () {
   FILE *fp;
   int elem;
   char nombre[20];

   printf("Introduzca el nombre del archivo a crear:");
   scanf("%s",nombre);

   /* Crea el archivo para escritura */
   if ((fp=fopen(nombre,"w")) != NULL) {
      printf("Introduzca los valores enteros positivos:");
      scanf("%d",elem);
```

```
            while (elem>=0) {
               /*Escribe el elemento en el archivo*/
               fprintf(fp,"%d",elem);
               scanf("%d",elem);
            }
            fclose(fp);   /*Cierra el archivo por seguridad*/
         }
         else
            printf("No se pudo crear el archivo");
      }
```

Ejercicios:

1.- Escriba una instrucción de entrada y salida que resuelva cada uno de los siguientes planteamientos:

a) Imprimir `9.375%`
b) Imprimir el primer caracter de la cadena constante "Monday"
c) Imprimir un string entre comillas
d) Imprimir `1234` justificado a la derecha en un campo de 10 dígitos
e) Imprimir `123.456789` en notación exponencial con signo y tres dígitos de precisión
f) Leer un valor double a la variable **`number`**
g) Imprimir `100` en forma octal precedida por `0`
h) Leer una cadena de caracteres al arreglo **`str`**
i) Leer un valor de la forma `3.5%`. Almacene el porcentaje en la variable **`float porc`** y elimine el signo `%`
j) Imprimir `3.333333` como un valor `long double` con signo, en un campo de 20 caracteres con una precisión de 3

2.- Diga que imprime cada una de las siguientes instrucciones. Si alguna es incorrecta diga por qué.

a) `printf("%[^0123456789]",n);`
b) `printf("%-10d\n",10000);`
c) `printf("%c\n","Esto es un string");`
d) `printf("%*.*lf\n",8,3,1024.987654);`
e) `printf("%#o\n%#X\n%#e\n",17,17,1008.83689);`
f) `printf("% ld\n%+ld\n\n",1000000,1000000);`
g) `printf("%10.2E\n",444.93738);`
h) `printf("%10.2g\n",444.93738);`
i) `printf("%d\n",10.987);`

3.- Escriba un programa que cargue el arreglo `number` de 10 elementos con enteros al azar, desde `1` hasta `1000`. Para cada uno de los valores, imprima el valor y el total acumulado del número de caracteres impresos. Utilice la especificación de conversión `%n` para determinar el número de caracteres impresos para cada valor. La salida debe tener el siguiente formato:

Total Caracteres	Valor
3	342
7	1000
10	963
11	6

4.- Escriba un programa para probar la diferencia entre los especificadores de conversión %d y %i al ser utilizados en enunciados **`scanf`**. Utilice las instrucciones `scanf("%i%d",&x,&y);` `printf("%d%d\n",x,y);` para introducir e imprimir los valores. Pruebe el programa con los siguientes conjuntos de datos de entrada: `10 10, -10 -10, 010 010, 0x10 0x10`.

5.- Indique cuáles de los siguientes enunciados son verdaderos y cuáles son falsos:

a) La función **`fscanf`** no puede ser utilizada para leer datos de la entrada estándar.
b) El programador debe utilizar **`fopen`** explícitamente, para abrir los flujos de entrada estándar, salida estándar y error estándar.
c) Para cerrar un archivo un programa debe llamar en forma explícita a la función **`fclose`**.
d) La función **`fprintf`** puede escribir a la salida estándar.
e) La función **`fseek`** puede buscar únicamente en relación con el principio de un archivo.

6.- Encuentre el error en cada una de las siguientes trozos de programa y diga cómo corregirlo

a)
```
{
     FOPEN *fPtr;

     fprintf(fPtr,"%d%s%s\n",cuenta,compañía,cantidad);
}
```
b)
```
{open("receive.dat","r+"); }
```
c) El archivo "tools.dat" debería ser abierto para añadir datos al archivo, sin descartar los datos actuales: `if ((tfPtr = fopen("tools.dat","w")) != NULL)`
d) El archivo "courses.dat" debería ser abierto para agregar sin modificar el contenido actual.
```
if ((tfPtr = fopen("courses.dat","w+")) != NULL)
```

7.- Suponga que la siguiente estructura ha sido definida y que el archivo está abierto para escritura.
```
struct persona{
     char apellido[15];
     char nombre[15];
     char edad[2];
}
```
Escriba instrucciones que resuelvan las siguientes proposiciones:

a) Inicialice el archivo "NOMBRES.DAT" de tal forma que existan 100 registros con apellido = "noasignado", nombre="" y edad="0'.
b) Lea de la entrada 10 apellidos, nombres y edades y escríbalos al archivo.

c) Actualice un registro. Si no existe información en el registro, indique al usuario "No info".
d) Borre un registro que tenga información mediante la reinicialización de dicho registro en particular.

8.- Hacer un subprograma que dados los nombres de dos archivos de caracteres haga una mezcla de los dos en uno nuevo archivo, con la condición de que se intercalen las líneas. El nombre del nuevo archivo también es dado como parámetro.

9.- Se tiene un archivo cuyos registros poseen la siguiente estructura: Nombre, Edad, Sexo. Se desea elaborar un programa que liste todas las mujeres que sean mayores de una edad X. Dichos datos se encuentran dentro del archivo. El nombre del archivo es dado por el usuario.

10.- Elabore un programa que cree un archivo de caracteres con líneas de tamaño fijo 80, a partir de un archivo (también de texto) que contiene líneas de tamaño variable entre 1 y 80 caracteres. Para ello el archivo creado se rellena de blancos en las líneas con tamaño menor que 80.

11.- Elabore un programa que tome todos los múltiplos de 4 hasta un N dado, y los almacene en un archivo de caracteres .

12.- Se tiene un archivo de enteros. Se desea elaborar un programa que nos indique la posición del máximo y la del mínimo de dicha lista de enteros.

13.- Se desea hacer una copia de un archivo de caracteres con la condición de que sólo se almacenen (en un nuevo archivo) las líneas que se encuentran en posición par.

14.- Tomar dos archivos de texto y hacer una mezcla de los dos en un nuevo archivo, con la condición de que se intercalen las líneas.

15.- Se tiene un archivo cuya estructura es la siguiente: `Nombre`, `Edad`, `Sexo`. Se desea elaborar un programa que liste todas las mujeres que sean mayores de una edad `X`. Dichos datos se encuentran dentro del archivo.

16.- Elaborar un programa que nos permita guardar el `CÓDIGO`, `NOMBRE` y `NOTA` de las materias cursadas, en un archivo. Las operaciones a realizar sobre dicho archivo son: listar las materias que pertenezcan a cierto departamento (Ej: `EC`, `MA`, `CI`, etc), listar las materias en las cuales se haya obtenido determinada nota, y listar todas las materias cursadas.

17.- Escribir un programa que reciba como entrada el nombre de un archivo y cree un archivo copia.dat con el contenido del mismo.

18.- Los datos mensuales que maneja un sistema de nómina están almacenados en un archivo de texto de la siguiente manera:

NombreEmpleado	HorasTrabajadas	SueldoPorHora
Pepe Rodríguez	5	3000
Julio López	15	100
..	..	...

El archivo se llama “master.dat”. Escribir un programa que calcule el promedio de horas trabajadas en el mes y la cantidad de dinero total a pagar en el mes.

19.- Con el mismo formato de archivo del problema anterior el gerente quiere obtener un archivo “master2.dat” que contenga solamente los empleados que hayan trabajado más de 20 horas y ganen menos de 1500 Bs. por hora.

20.- El sistema de control de inscripciones de D.A.C.E mantiene la información de los estudiantes en un archivo de texto de la siguiente manera:

CarnetEst	NombreEst	IndiceEst	SexoEst
90-22250	Ricado P.	3.61	M
90-22251	María P.	4.23	F
...	...	...	...

El archivo se llama “CEST.DAT”. Escribir un programa que:

a) Calcule el índice promedio de los estudiantes.
b) Calcule el número de estudiantes con carnet menor a 91.
c) Calcule el número de estudiantes masculinos con índice superior a 3.5.
d) Dado un número de carnet muestre el índice del estudiante.

21.- Con el formato de archivo anterior bienestar estudiantil desea obtener “CEST1.DAT”, “CEST2.DAT”, “CEST3.DAT”, “CEST4.DAT” tales que:

a) “CEST1.DAT” contenga la información de los estudiantes masculinos carnet 90 con índice mayor o igual a 3.50
b) “CEST2.DAT” contenga la información de los estudiantes con índice mayor a 4.25 que entraron después del año 92
c) “CEST3.DAT” contenga los nombres de las estudiantes 94 cuyos apellidos comiencen con A y B.
d) “CEST4.DAT” contenga los carnet de los estudiantes que se llamen José y tengan índice superior a 4.00