



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:

Edgar Tista García

Asignatura:

Estructuras de Datos y Algoritmos I

Grupo:

1

No. de Práctica(s):

1

Integrante(s):

Ugalde Velasco Armando

*No. de Equipo de
cómputo empleado:*

47

No. de Lista:

Semestre:

2020-2

Fecha de entrega:

12 de febrero de 2020

Observaciones:

CALIFICACIÓN: _____

OBJETIVO

Utilizar arreglos unidimensionales y multidimensionales para dar solución a problemas computacionales.

EJEMPLO DE LA GUÍA DE LABORATORIO

La escítala espartana fue uno de los primeros métodos criptográficos conocidos. El método consiste en enrollar una tira de escritura a lo largo de un palo y escribir sobre la tira una vez enrollada.



La implementación del código presente en la guía es la siguiente:

```
*** ESC=TALA ESPARTANA ***
¿Qué desea realizar?
1) Crear mensaje cifrado.
2) Descifrar mensaje.
3) Salir.
```

En esta captura de pantalla podemos observar el menú principal del programa, el cual cuenta con tres opciones: crear mensaje cifrado, descifrar mensaje y salir.

Podemos notar que algunos caracteres no se muestran correctamente, como la “í” y el signo “¿”. Esto es debido a la representación de texto utilizada. En este caso, el estándar

utilizado es ASCII, el cual no contiene a los signos antes mencionados.

```
*** ESC=TALA ESPARTANA ***
¿Qué desea realizar?
1) Crear mensaje cifrado.
2) Descifrar mensaje.
3) Salir.
1
Ingresar el tamaño de la escítala:
Renglones:3
Columnas:4
Escriba el texto a cifrar:
holavacalola
El texto en la tira queda de la siguiente manera:
hvloaolclaaa

*** ESC=TALA ESPARTANA ***
¿Qué desea realizar?
1) Crear mensaje cifrado.
2) Descifrar mensaje.
3) Salir.
-
```

En esta captura de pantalla, se eligió la opción 1 del menú: crear un mensaje cifrado. Se introdujeron la cantidad de renglones y columnas de la escítala, que fueron 3 y 4, respectivamente. Posteriormente se introdujo el texto a cifrar, el cual fue “holavacalola”.

La longitud de este texto es de doce letras, lo cual facilita la representación en un arreglo de 3x4, como se muestra a continuación.

```
h o l a
v a c a
l o l a
```

En el programa, el texto introducido se almacena de esta forma: el texto se almacena en un arreglo de caracteres de dimensión **renglones x columnas**. Después se crea un arreglo de dos dimensiones, con los renglones y columnas introducidos, y los caracteres se guardan en cada renglón, como se mostró anteriormente. Finalmente, el texto cifrado se obtiene al invertir las columnas y renglones del arreglo, como se muestra a continuación.

```
h v l
o a o
l c l
a a a
```


En la captura de pantalla, podemos observar que el texto cifrado es “hvloaolclaaa”, lo cual coincide con el contenido del arreglo anterior. En el programa no se guarda un segundo

arreglo, únicamente se imprime el contenido de éste, invirtiendo las columnas y los renglones.

Lo anterior se realiza mediante el uso de dos ciclos **for**. En el primero se imprime cada renglón del arreglo y en el segundo se imprime cada columna.

```
for (i=0 ; i<ren ; i++)
{
    for (j=0 ; j<col ; j++)
    {
        escitala[i][j] = texto[k++];
    }
}
printf("El texto en la tira queda de la siguiente manera:\n");
for (i=0 ; i<col ; i++)
{
    for (j=0 ; j<ren ; j++)
    {
        printf("%c", escitala[j][i]);
    }
}
```

Ahora, descifraremos el mensaje anterior.



```
Renglones:3
Columnas:4
Escriba el texto a cifrar:
holavacalola
El texto en la tira queda de la siguiente manera:
hvloaolclaaa

*** ESC=TALA ESPARTANA ***
¿Qué desea realizar?
1) Crear mensaje cifrado.
2) Descifrar mensaje.
3) Salir.
2
Ingresar el tamaño de la escítala:
Renglones:3
Columnas:4
Escriba el texto a descifrar:
hvloaolclaaa
El texto descifrado es:
holavacalola
*** ESC=TALA ESPARTANA ***
¿Qué desea realizar?
1) Crear mensaje cifrado.
2) Descifrar mensaje.
3) Salir.
```

Al escoger la opción 2 para descifrar el mensaje, nos es solicitado el número de renglones, columnas y el texto para obtener el mensaje descifrado. Introducimos los datos, los cuales ya conocemos: 3, 4 y “hvloaolclaaa”, respectivamente.

Podemos observar que el mensaje descifrado es correcto: “holavacalola”.

En el programa, al igual que en la opción 1, se crea un arreglo bidimensional con las dimensiones especificadas, y un arreglo unidimensional de caracteres de longitud **ren glones x columnas**.

Posteriormente, el texto es copiado al arreglo bidimensional, de tal forma que se copia columna por columna y no renglón por renglón. Lo anterior lo podemos observar en el siguiente ciclo:

```
for (i=0 ; i<col ; i++)
{
    for (j=0 ; j<ren ; j++)
    {
        escitala[j][i] = texto[k++];
    }
}
```

Al realizar lo anterior, tenemos ahora el arreglo de la escítala original, por lo que podemos imprimir cada renglón de éste, resultando en el mensaje inicial.

EJERCICIO 1

El programa compila y ejecuta correctamente. Su salida es la siguiente:

```
[Granada47:Desktop alumno$ ./ejercicio1
Arreglo [0][0][0]: 2
Arreglo [0][0][1]: 4
Arreglo [0][0][2]: 6
Arreglo [0][1][0]: 12
Arreglo [0][1][1]: 14
Arreglo [0][1][2]: 16
Arreglo [1][0][0]: 22
Arreglo [1][0][1]: 24
Arreglo [1][0][2]: 26
Arreglo [1][1][0]: 32
Arreglo [1][1][1]: 34
Arreglo [1][1][2]: 36
Arreglo [2][0][0]: 42
Arreglo [2][0][1]: 44
Arreglo [2][0][2]: 46
Arreglo [2][1][0]: 52
Arreglo [2][1][1]: 54
Arreglo [2][1][2]: 56
Arreglo [3][0][0]: 1321926718
Arreglo [3][0][1]: 1722900698
Arreglo [3][0][2]: -376046816
Arreglo [3][1][0]: 32767
Arreglo [3][1][1]: 0
Arreglo [3][1][2]: 0
Arreglo [4][0][0]: -376046512
Arreglo [4][0][1]: 32766
Arreglo [4][0][2]: 0
Arreglo [4][1][0]: 32766
Arreglo [4][1][1]: -376046471
Arreglo [4][1][2]: 32766
Granada47:Desktop alumno$ █
```

Sin embargo, podemos notar que los valores mostrados son incorrectos. La representación de nuestro arreglo original es la siguiente:

Primer plano	2	4	6	8	10
	12	14	16	18	20

Segundo plano	22	24	26	28	30
	32	34	36	38	40

Tercer plano	42	44	46	48	50
	52	54	56	58	60

```
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 2; j++)
    {
        for (k = 0; k < 3; k++)
        {
            printf("Arreglo [%d][%d][%d]: %d \n", i, j, k, arr[i][j][k]);
        }
    }
}
```

En el ciclo accedemos al arreglo en posiciones erróneas: el contador k del ciclo interior indica las columnas. Sin embargo, éste toma los valores de 0 a 2. Sabemos que la longitud de las columnas es 5, por lo que podemos concluir que únicamente se accede a los primeros 3 elementos de éstas, ignorando los últimos 2.

Lo anterior lo podemos observar claramente en la salida del programa para el plano 1: se imprime 2, 4 y 6 (los primeros 3 elementos de la fila 1), y después se imprime 12, 14 y 16 (los primeros 3 elementos de la fila 2). Nos podemos percatar de que los números 8, 10, 18 y 20 no son impresos. En consecuencia, podemos concluir que **este contador debería**

de tomar cinco valores para lograr imprimir todos los elementos posicionados en las columnas de cada fila: **0, 1, 2, 3 y 4**.

El contador **j**, que indica las filas, es correcto, ya que sus valores pueden ser dos: 0 y 1, y sabemos que cada plano tiene dos filas.

Sin embargo, el contador **i** es el problema del programa, ya que puede tomar cinco valores: de 0 a 4, pero el arreglo no cuenta con 5 planos, sino 3. Por lo tanto, los índices válidos son 0, 1 y 2, y, en consecuencia, al acceder al índice 3 o 4 el programa arroja valores inesperados. Lo anterior ocurre debido a que estamos accediendo a localidades de memoria que no corresponden al arreglo, por lo que se produce un comportamiento indefinido. En consecuencia, podemos concluir que **este contador debería de tomar sólo tres valores: 0, 1 y 2**.

Ahora bien, al realizar la corrección de los respectivos ciclos, tenemos que el contador **i** en realidad debe de tomar los **valores menores a 3**, el contador **j** debe **permanecer igual**, y el contador **k** debe de tomar los **valores menores a 5**, como ya se mencionó. El ciclo corregido es el siguiente:

```
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 2; j++)
    {
        for (k = 0; k < 5; k++)
        {
            printf("Arreglo [%d][%d][%d]: %d \n", i, j, k, arr[i][j][k]);
        }
    }
}
```

Como podemos observar, los ciclos anidados corresponden con las dimensiones del arreglo: **3 planos, 2 filas y 5 columnas**. En consecuencia, la salida del programa ahora es la esperada.

```

Arreglo [0][0][0]: 2
Arreglo [0][0][1]: 4
Arreglo [0][0][2]: 6
Arreglo [0][0][3]: 8
Arreglo [0][0][4]: 10
Arreglo [0][1][0]: 12
Arreglo [0][1][1]: 14
Arreglo [0][1][2]: 16
Arreglo [0][1][3]: 18
Arreglo [0][1][4]: 20
Arreglo [1][0][0]: 22
Arreglo [1][0][1]: 24
Arreglo [1][0][2]: 26
Arreglo [1][0][3]: 28
Arreglo [1][0][4]: 30
Arreglo [1][1][0]: 32
Arreglo [1][1][1]: 34
Arreglo [1][1][2]: 36
Arreglo [1][1][3]: 38
Arreglo [1][1][4]: 40
Arreglo [2][0][0]: 42
Arreglo [2][0][1]: 44
Arreglo [2][0][2]: 46
Arreglo [2][0][3]: 48
Arreglo [2][0][4]: 50
Arreglo [2][1][0]: 52
Arreglo [2][1][1]: 54
Arreglo [2][1][2]: 56
Arreglo [2][1][3]: 58
Arreglo [2][1][4]: 60

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.

```

EJERCICIO 2

El programa realizado cumple con la funcionalidad solicitada. Su funcionamiento es el siguiente:

Primero, se solicitan 15 números enteros al usuario:

```

[Granada47:Desktop alumno$ ./ejercicio2
Introduzca el valor 1
2
Introduzca el valor 2
4
Introduzca el valor 3
3
Introduzca el valor 4
5
Introduzca el valor 5
53
Introduzca el valor 6
2
Introduzca el valor 7
3
Introduzca el valor 8
4
Introduzca el valor 9
5
Introduzca el valor 10
5
Introduzca el valor 11
5
Introduzca el valor 12
5
Introduzca el valor 13
5
Introduzca el valor 14
5
Introduzca el valor 15
2

```


En el programa, se crea un arreglo unidimensional de 15 elementos y después se almacenan los datos en éste mediante un ciclo **for**, solicitando al usuario que introduzca los datos.

```
int arr[15];

for (int i = 0; i < 15; i++)
{
    printf("Introduzca el valor %d\n", i+1);
    scanf("%d", &arr[i]);
}
```

Luego, el usuario debe elegir una opción, ingresando un número del 1 al 4. Si el usuario no introduce un número válido se le será solicitado de nuevo, mediante la utilización de un ciclo **while**.

```
Elija una opción:
1) Suma
2) Multiplicacion
3) Suma de elementos divisibles entre 4
4) Multiplicar por 3 cada elemento
1
```

Después, se imprimirá en pantalla el resultado deseado. Lo anterior se implementó mediante una estructura **switch**, y 4 funciones auxiliares que realizaran las operaciones especificadas. El argumento de estas funciones es el arreglo almacenado con los números introducidos por el usuario.

En cada una de ellas se imprime el arreglo original al inicio. Esto se lleva a cabo con una función auxiliar: **imprimirOriginal**.

Posteriormente, se imprime el resultado deseado. Podemos observar que los resultados son correctos en cada opción.

Opción 1: Suma

En este caso, la función auxiliar utiliza un ciclo **for** para acceder a todos los elementos del arreglo y sumarlos a una variable creada llamada **suma**, mediante el operador de asignación **+=**.

```
El arreglo original es:
2 4 5 7 3 2 2 2 22 2 2 22 2 2 3
La suma es: 82
```

Opción 2: Multiplicación

La función auxiliar también utiliza un ciclo **for** para acceder a todos los elementos del arreglo, pero en este caso se utiliza el operador de asignación ***=** para multiplicarlos por una variable acumuladora llamada **multi**, cuyo valor inicial es 1.

```
El arreglo original es:
1 2 4 5 1 3 4 1 2 34 2 2 3 4 5
La multiplicacion de los elementos es: 198048688
```

Opción 3: Suma de elementos divisibles entre 4

En este caso se realiza el mismo procedimiento que en la opción 1, pero se agrega una condición al momento de sumar el elemento: únicamente se agrega a la suma si éste es múltiplo de 4, o bien, si su módulo respecto a 4 es cero.

```
El arreglo original es:
2 3 455 6 5 23 23 23 23432 42 423 234 423 423 432
La suma de los elementos divisibles entre 4 es: 23864
```

Opción 4: Multiplicar por 3 cada elemento

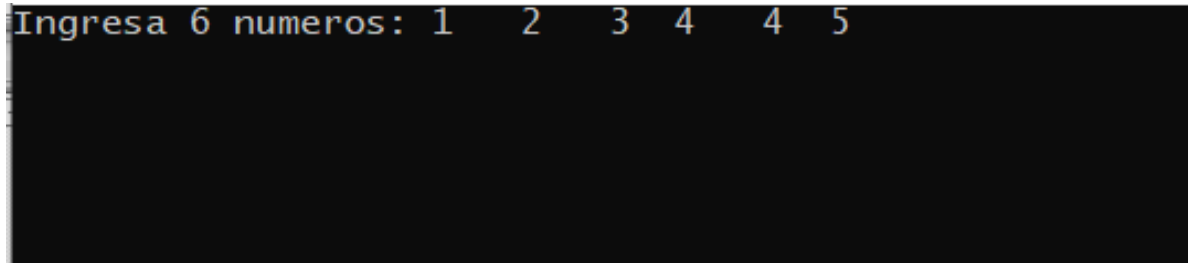
En este caso únicamente se imprime el valor de cada elemento del arreglo multiplicado por 3, sin modificar al original, ya que la obtención del valor se realiza en la función **printf**. Al igual que en las otras opciones se utiliza un ciclo **for** para acceder a cada elemento del arreglo.

```
El arreglo original es:
21 3 4 5 5 2 3 3 3 34 5 5 4 32 1
Los elementos multiplicados por 3 son:
63 9 12 15 15 6 9 9 9 102 15 15 12 96 3
```

EJERCICIO 3

El programa realizado cumple con la funcionalidad solicitada. Su funcionamiento es el siguiente:

Se le solicita al usuario que ingrese 6 números:



```
Ingresa 6 numeros: 1 2 3 4 4 5
```

Estos pueden estar separados por espacios o por renglones. Lo anterior se lleva a cabo mediante la creación de un arreglo de 4 renglones y 6 columnas, y la utilización de la función **scanf**, como se muestra en la siguiente imagen:

```
int arr[4][6], i, j;

printf("Ingresa 6 numeros: ");
// Se almacenan los datos proporcionados en la primera fila del arreglo.
scanf("%d%d%d%d%d%d", &arr[0][0], &arr[0][1], &arr[0][2], &arr[0][3], &arr[0][4], &arr[0][5]);
```

Como podemos observar, los datos proporcionados se almacenan en la primera fila del arreglo. Posteriormente, se almacenan los valores de las filas restantes por medio de un ciclo **for**, el cual multiplica por 3 cada valor de la fila anterior y lo almacena en la fila actual en el contador.

```
for (i = 1; i < 4; i++)
{
    for (j = 0; j < 6; j++)
    {
        arr[i][j] = arr[i-1][j] * 3;
    }
}
```

Finalmente, se imprime el arreglo por medio de otro ciclo **for**.

```
Ingresa 6 numeros: 1 2 3 4 5 6
El nuevo arreglo es:
1 2 3 4 5 6
3 6 9 12 15 18
9 18 27 36 45 54
27 54 81 108 135 162
```

Podemos percatarnos de que los elementos en cada columna aumentan el triple en cada fila, lo cual es la salida esperada.

3.1 Para resolver el problema respecto al almacenamiento de números grandes, podríamos implementar un arreglo con el tipo de dato ***long long int***. Sin embargo, éste sólo almacenaría enteros de 8 bytes, con el rango de -2^{63} a $2^{63}-1$. Como una solución adicional, podríamos implementar un tipo de dato abstracto que incluyera un arreglo unidimensional con el tamaño apropiado para cada número, y así lograr almacenar cualquier número independientemente de su tamaño.

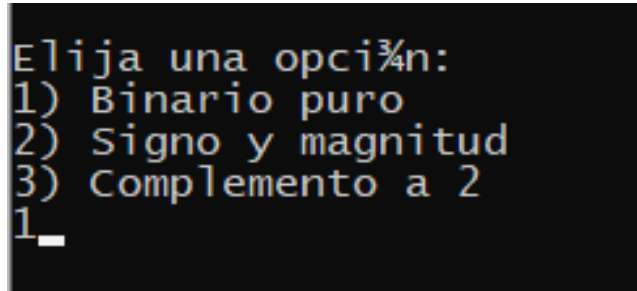
EJERCICIO 4

El programa realizado cumple con la funcionalidad solicitada. Su funcionamiento es el siguiente:

Primero, se le solicita al usuario que introduzca 10 bits, individualmente, los cuales son almacenados en un arreglo unidimensional de tamaño 10. Si el valor de alguno de ellos es diferente de 0 o 1, éste se vuelve a solicitar.

```
Introduzca el bit 1
1
Introduzca el bit 2
0
Introduzca el bit 3
0
Introduzca el bit 4
1
Introduzca el bit 5
0
Introduzca el bit 6
0
Introduzca el bit 7
0
Introduzca el bit 8
1
Introduzca el bit 9
1
Introduzca el bit 10
1
```

Luego, se solicita al usuario que introduzca el número de la codificación deseada: uno para **binario puro**, dos para **signo y magnitud**, y tres para **complemento a dos**. Si el usuario introduce un valor erróneo se le solicita que lo introduzca de nuevo.



Finalmente, se ejecuta la conversión deseada. Podemos observar la implementación y salida en cada caso:

Binario puro

```
void puro(int arr[])
{
    int numero, i;
    numero = 0;

    int potencia = 9;
    for (i = 0; i < 10; i++)
    {
        if (arr[i])
        {
            numero += pow(2, potencia);
        }
        potencia--;
    }

    printf("El valor es: %d", numero);
}
```

En este caso, la conversión se realiza haciendo la suma de cada dígito del número, tomando en cuenta su valor real en decimal. Es decir, se inicia con el primer dígito, cuyo valor decimal es 2^9 . Por lo tanto, la variable **potencia** inicia con el valor 9, y, conforme el ciclo avanza en los elementos del arreglo, el valor de la potencia disminuye una unidad, hasta finalmente llegar a 0.

Únicamente se acumulan los valores cuyo bit es 1, lo cual se logra mediante la sentencia **if**: si el valor en el arreglo es 1, éste será evaluado como verdadero, de lo contrario es 0, por lo que será evaluado como falso.

```

Introduzca el bit 1
1
Introduzca el bit 2
0
Introduzca el bit 3
0
Introduzca el bit 4
1
Introduzca el bit 5
0
Introduzca el bit 6
0
Introduzca el bit 7
0
Introduzca el bit 8
1
Introduzca el bit 9
1
Introduzca el bit 10
1

Elija una opción:
1) Binario puro
2) Signo y magnitud
3) Complemento a 2
11
Digite un valor valido

Elija una opción:
1) Binario puro
2) Signo y magnitud
3) Complemento a 2
1
El valor es: 583

```

Signo y magnitud

```

void signoyMagnitud(int arr[])
{
    int magnitud, i;
    magnitud = 0;

    int potencia = 8;
    for (i = 1; i < 10; i++)
    {
        if (arr[i])
        {
            magnitud += pow(2, potencia);
        }
        potencia--;
    }

    printf("El valor es: %s%d", arr[0]? "-" : "", magnitud);
}

```

En este caso, la conversión se realiza de forma similar a la anterior. La única diferencia es que se inicia desde la posición 1 en el arreglo, como podemos observar en el ciclo: el **valor inicial** de la variable **i** es **1**. De esta forma se calcula la magnitud del número y se almacena en una variable del mismo nombre.

Finalmente, al momento de imprimir el resultado, se utilizan dos valores para definir el número: el signo y la magnitud. La magnitud se imprime como un tipo de dato entero en base decimal, y el signo se imprime utilizando el **operador ternario**: si el primer bit en el

arreglo es 1, entonces el número es negativo, por lo tanto, la expresión se evalúa al carácter “-” en una cadena, de lo contrario la expresión se evalúa a una cadena vacía.

```
Introduzca el bit 1
1
Introduzca el bit 2
0
Introduzca el bit 3
0
Introduzca el bit 4
1
Introduzca el bit 5
0
Introduzca el bit 6
0
Introduzca el bit 7
1
Introduzca el bit 8
1
Introduzca el bit 9
1
Introduzca el bit 10
1

Elija una opción:
1) Binario puro
2) Signo y magnitud
3) Complemento a 2
2
El valor es: -79
Process returned 0 (0x0)   execution time : 6.980 s
Press any key to continue.
```

Complemento a 2

```
void compA2(int arr[])
{
    if (arr[0])
    {
        int i;
        int encontrado = 0;

        for (i = 9; i > 0; i--)
        {
            if (!encontrado)
            {
                if (arr[i])
                {
                    encontrado = 1;
                }
            }
            else
            {
                arr[i] = arr[i] ? 0 : 1;
            }
        }
    }

    signoyMagnitud(arr);
}
```

En este caso, si el número es positivo, únicamente se toma como una representación común de **magnitud y signo**, lo cual podemos observar claramente al final de la función, donde en cualquier caso se ejecuta la conversión mencionada.

Sin embargo, si el número es negativo, es necesario obtener su **complemento a 2**. Dicha tarea se realiza implementando el algoritmo común de conversión: copiar todos los dígitos hasta encontrar el primer dígito cuyo valor sea 1, de derecha a izquierda e incluyendo a éste, y finalmente invertir los dígitos restantes. En el programa no se realiza ninguna copia del arreglo, únicamente se modifican los bits pertinentes en el arreglo existente.

Dicha tarea se realiza en el ciclo **for**, donde se utilizó la variable **encontrado** para que fungiera como una bandera.

El flujo del programa es el siguiente: el ciclo inicia desde 9, es decir, desde el dígito de la **extrema derecha**, y termina en 1, la posición del **primer dígito** de la magnitud. Luego, en la sentencia **if**, se comprueba si se ha encontrado el primer 1. Si no es así, no se modifica el arreglo, y se comprueba si el valor actual analizado es 1, para cambiar el valor de la bandera. Si es así, se actualiza el valor en el arreglo al valor opuesto.

Finalmente, se ejecuta la conversión con signo y magnitud.

```
Introduzca el bit 1
1
Introduzca el bit 2
1
Introduzca el bit 3
1
Introduzca el bit 4
0
Introduzca el bit 5
0
Introduzca el bit 6
0
Introduzca el bit 7
0
Introduzca el bit 8
1
Introduzca el bit 9
0
Introduzca el bit 10
1

Elija una opción:
1) Binario puro
2) Signo y magnitud
3) Complemento a 2
3
El valor es: -123
Process returned 0 (0x0)   execution time : 8.066 s
Press any key to continue.
```

CONCLUSIONES

Podemos decir que se cumplió el objetivo planteado al inicio de la práctica, ya que se resolvieron satisfactoriamente todos los ejercicios propuestos por medio de la implementación de arreglos de una o varias dimensiones, el cual era el punto principal. Además, logré identificar los errores potenciales en su utilización, así como la forma de

resolverlos, lo cual es una destreza fundamental en la programación. Un claro ejemplo de lo anterior ocurrió en el ejercicio 1, donde fue necesario corregir el programa dado.

Los ejercicios propuestos me parecieron ideales para reafirmar la práctica de este concepto y otros relevantes, como la conversión de números binarios presente en el ejercicio 4.