

# Tarea 3. Programación y Algoritmos

Carlos Giovanni Encinia González

31 agosto 2021

## Ejercicio 1

Programa que verifique si un número de entrada dado es palíndromo. Un palíndromo, es un número o string que se lee igual en un sentido que en otro (por ejemplo: Ana, 121...). Usar una función que regrese un numero invertido (al revés).

Firma de la función: `int invert(int number)`

`Invert(123)` → regresa 321

Impresión en terminal: “El número ### es (o no es) palíndromo”

**NOTA:** No usar recursión ni arreglos.

Figure 1: Es palindromo

```
Ingresa un numero positivo
14541

El numero es 14541
El numero invertido es 14541
14541 es palindromo
```

En la figura (2) el usuario agrego un numero negativo, este numero es invertido, si un palindromo es ingresado el programa dira que es palindromo aunque sea negativo, unicamente considerara los digitos para dar un veredicto.

## Respuesta

El algoritmo pide al usuario que ingrese un entero, al inicio sugiere que el entero sea positivo, aunque el usuario puede agregar uno negativo. Tal y como lo pide el ejercicio el algoritmo imprime en pantalla si el numero dado es un palíndromo o no, además de imprimir el numero original y el numero invertido.

En el codigo existen tres funciones, una llamada `potencia()` que se usa dentro de una funcion llamada `invierte_numero()`, y la funcion `main()`, la funcion que hace la magia es la funcion `invierte_numero()`, la cual descompone el numero dado en varios digitos y dependiendo de la posicion original del digito este es transformado a la posicion que le tocara si se invierte el numero, para realizar esto al principio se debe de calcular el orden del numero que se esta leyendo, por ejemplo el numero 300 es de un orden de 2, y con esto me refiero que es necesario dividir entre  $10^2$  para obtener el ultimo digito. En seguida se muestran algunos resultados obtenidos con la ejecucion del programa.

En la figura (1) podemos observar que el numero dado es un palíndromo.

Figure 2: Negativos

```
Ingresa un numero positivo
-1234

El numero es -1234
El numero invertido es -4321
No es palindromo u.u
```

Si al final del numero existen ceros figura (3), este cero no se vera reflejado en el numero invertido ya que los datos tipo numero no llevan un cero por delante.

Figure 3: Ceros al final

```
Ingresa un numero positivo
1900010

El numero es 1900010
El numero invertido es 100091
No es palindromo u.u
```

---

---

Code 1: main.c

---

---

```
1 //Giovanny Encinia
  //30/08/2021
3 #include <stdio.h>
  #define TEN 10
5 #define ONE 1
  #define ZERO 0
7
  long potencia(int number, int n)
9 {
    /*Una funcion para efectuar la potencia de un numero
11    unicamente se le da el numero y la potencia a elevar*/

13    int i = ZERO;
    long result = ONE;

15    while(i < n)
17    {
        result *= number;
19        i++;
    }

21    return result;
23 }

25 long invierte_numero(long n_number)
  {
27
    long resultado, ten, sum = ZERO;
29    int i = ZERO, j = ZERO;

31    ten = TEN;

33    //encuentra el numero de veces que debe ser dividido
    //por diez para poder llegar a una unidad con el primer digito
35    resultado = n_number / ten;

37    while(resultado != ZERO)
    {
39        // el ultimo evento es cuando ten hace que halla decimales
        ten *= TEN;
41        resultado = n_number / ten;
        i++;
43    }

45    ten /= TEN; // regresa a cuando ten deja una unidad
47    while(j <= i)
```

```

49     {
50         resultado = n_number / ten; // obtener los digitos
51         // n_number es el resto despues de obtener el digito
52         // ahora ten debe ser menor ej. antes eran centenas,
53         //ahora decenas las que se obtendran
54         n_number %= ten;
55
56         ten /= TEN;
57         //cada uno de los digitos multiplicalos por la potencia
58         //correspondiente de diez
59         //de tal manera que see vayan invirtiendo los numeros
60         sum += resultado * potencia(TEN, j++);
61     }
62
63     // al final solo nos falta agregar la unidad la cual sera
64     //el ultimo numero n_number
65     // que es el resultado del modulo del numero original entre
66     //la potencia mas grande que deja
67     // la unidad y asi se invierte el numero
68     sum += n_number * potencia(TEN, j);
69
70     return sum;
71 }
72
73 int main()
74 {
75     long number, invertido;
76
77     printf("\t\tIngrese un numero positivo\n");
78     scanf("%ld", &number);
79     invertido = invierte_numero(number);
80     printf("\n\tEl numero es %ld\n", number);
81     printf("\tEl numero invertido es %ld\n", invertido);
82
83     if(number == invertido)
84     {
85         printf("\t%ld es palindromo\n", number);
86     }
87     else
88     {
89         printf("\tNo es palindromo u.u\n");
90     }
91
92     return ZERO;
93 }

```

---

## Ejercicio 2

Escribe un programa que implemente la estructura de datos Stack usando un arreglo de enteros. Se puede asumir que la capacidad máxima del Stack es de 10 enteros. El arreglo se puede definir globalmente y se implementaran las funciones push y pop:

```
void push(int)
```

```
int pop( )
```

Simula 10 operaciones e imprime en terminal el estado interno del Stack.

## Respuesta

El algoritmo no necesita que el usuario introduzca datos, únicamente se muestra como cambia el estado de un stack al realizar ciertas operaciones. Las operaciones que están disponibles son `is_empty()`, `is_full()`, `push()` y `pop()`. La manera en que se trabaja el arreglo es usando dos variables globales muy importantes, que son los datos actuales dentro del stack denotados por  $n$  y el array donde se guarda el stack.

Si se quiere realizar un push pero el stack ya esta lleno, mandara un mensaje a la salida de datos standard diciendo que es imposible dicha operación en ese momento figura (6), algo similar sucede si se quiere hacer pop y el stack esta vacío. Tambien se agrego una función que imprime los datos del stack y el numero de datos que existen en el.

En las figuras (4) y (5) se muestran algunas operaciones realizadas para primero llenar el arreglo y después hacer pop a algunos elementos. Después de la imágenes se muestra el código del programa.

Figure 4: Lenando con push()

```
push al valor 0
El stack tiene 1 elementos:
0

push al valor 3
El stack tiene 2 elementos:
0 3

push al valor 6
El stack tiene 3 elementos:
0 3 6

push al valor 9
El stack tiene 4 elementos:
0 3 6 9

push al valor 12
El stack tiene 5 elementos:
0 3 6 9 12

push al valor 15
El stack tiene 6 elementos:
0 3 6 9 12 15

push al valor 18
El stack tiene 7 elementos:
0 3 6 9 12 15 18

push al valor 21
El stack tiene 8 elementos:
0 3 6 9 12 15 18 21

push al valor 24
El stack tiene 9 elementos:
0 3 6 9 12 15 18 21 24

push al valor 27
El stack tiene 10 elementos:
0 3 6 9 12 15 18 21 24 27

push al valor 30
El arreglo ya estalleno, imposible hacer push
El stack tiene 10 elementos:
0 3 6 9 12 15 18 21 24 27
```

Figure 5: Vaciando con pop()

```
pop al valor 27
El stack tiene 9 elementos:
0 3 6 9 12 15 18 21 24

pop al valor 24
pop al valor 21
pop al valor 18
El stack tiene 7 elementos:
0 3 6 9 12 15 11
```

Figure 6: Stack lleno

```
push al valor 30
El arreglo ya estalleno, imposible hacer push
El stack es:
0 3 6 9 12 15 18 21 24 27
```

---

---

Code 2: main.c

---

---

```
1 //Giovanny Encinia
  //30/08/2021
3 #include <stdio.h>
  #define ZERO 0
5 #define ONE 1
  #define SIZE 10
7 #define LEN(a) (sizeof(a) / sizeof(a[ZERO]))

9 void push(int);
  int pop(void);
11 void imprime(void);
  int is_full(void);
13 int is_empty(void);

15 int a[SIZE];
  int n = ZERO; // servira para controlar el stack
17
  int is_full(void)
19 {
    /*Comprueba que el stack esta vacio ,
21     esto con ayuda de la
    variable global n*/
23
    if(n == SIZE)
25     {
        return ONE; //verdadero
27     }
    else
29     {
        return ZERO; //falso
31     }
    }
33
  int is_empty(void)
35 {
    /*Comprueba que el stack este lleno ,
37     esto con ayuda de la
    variable global n*/
39
    if(n == ZERO)
41     {
        return ONE;
43     }
    else
45     {
        return ZERO;
47     }
    }
```

```

49 void push(int number)
50 {
51     if(!is_full())
52     {
53         a[n] = number;
54         n++; // actualiza el estado del stack
55     }
56     else
57     {
58         printf("\t\tEl arreglo ya esta\
59 lleno, imposible hacer push\n");
60     }
61 }
62
63 }
64
65 int pop(void)
66 {
67     int number;
68     if(!is_empty())
69     {
70         number = a[n - ONE];
71         n -= ONE; // actualiza el estado del stack
72     }
73     else
74     {
75         printf("\t\tEl arreglo ya esta\
76 vacio imposible hacer pop, regresara 0\n");
77         return ZERO;
78     }
79
80     return number;
81 }
82
83 void imprime(void)
84 {
85     int i = ZERO;
86
87     printf("\t\tEl stack tiene %d \
88 elementos:\n\t\t", n);
89
90     while(i < n)
91     {
92         printf("%d ", a[i]);
93         i++;
94     }
95
96     printf("\n\n");
97 }

```

```

99  int main(void)
    {
101      int i = ZERO;

103      while(i < SIZE + ONE)
      {
105          printf("\t\tpush al valor %d\n", i*3);
          push(i*3);
107          imprime();
          i++;
109      }

111      printf("\t\tpop al valor %d\n", pop());
      imprime();
113      printf("\t\tpop al valor %d\n", pop());
      printf("\t\tpop al valor %d\n", pop());
115      printf("\t\tpop al valor %d\n", pop());
      push(i++);
117      imprime();

119      return ZERO;

121  }

```

---

## Ejercicio 3

Escribe una función que reciba un arreglo de enteros y que calcule el producto acumulado para cada entrada usando

$$a[i] = \prod_{j \neq i} a[j]$$

Se espera que la complejidad del algoritmo sea lineal  $O(n)$ . Se puede asumir que la longitud máxima del arreglo será 50.

Firma de la función: `products(int a[], int products[])` Imprime en terminal el arreglo original y el arreglo de productos.

NOTA: El martes 31 de agosto estudiaremos funciones con argumentos de tipo arreglo.

## Respuesta

Se hicieron dos versiones del algoritmo, al principio creí que solamente se debería de obtener el producto acumulado del arreglo para cada elemento, por ejemplo para el arreglo  $[1, 2, 3, 4]$  el resultado sería  $[1, 2, 6, 12]$  figura (7). Entonces este programa que también sera entregado, pide al usuario la cantidad de elementos que tendrá el arreglo (máximo de 50) y comienza a realizar el producto como se mostró anteriormente. Al final imprime el arreglo entregado y el arreglo con el producto acumulado.

La segunda versión al final me hizo mas sentido ya que mencionaba que  $i \neq j$ , esta versión también pide el numero de elementos del arreglo y después empieza a leer cada elemento, al terminar multiplica todo los elementos exceptuando el elemento cuyo índice en el arreglo original es el mismo que en el que se guardara. Se toman tres casos distintos, el primer caso es donde todos los elementos son distintos de cero figura (8), el segundo es donde solo existe un cero figura (9), y el tercero es donde hay mas de un cero figura (10). Cuando existe un cero, el índice donde esta el elemento cero tendrá el valor de la multiplicación de cada uno de los elementos, y los demás espacios en el arreglo serán cero. El primer caso es trivial y en el segundo caso todos los elementos son ceros.

Figure 7: Primera versión

```
Cuantos elementos tendra el arreglo (maximo 50)?
5
Agregue el elemento 0
1
Agregue el elemento 1
2
Agregue el elemento 2
3
Agregue el elemento 3
4
Agregue el elemento 4
5

El arreglo original es:
1 2 3 4 5

El arreglo con producto acumulado es:
1 2 6 24 120
```

Figure 8: Segunda versión

```
Ingrese el numero de elementos maximo 50
5
Ingrese el elemento 0
1
Ingrese el elemento 1
2
Ingrese el elemento 2
3
Ingrese el elemento 3
4
Ingrese el elemento 4
5

El arreglo original es:
1 2 3 4 5

El arreglo con oroductos es:
120 60 40 30 24
```



Figure 9: Segunda versión un cero

```
Ingrese el numero de elementos maximo 50
5
Ingrese el elemento 0
0
Ingrese el elemento 1
2
Ingrese el elemento 2
3
Ingrese el elemento 3
4
Ingrese el elemento 4
5

El arreglo original es:
0 2 3 4 5

El arreglo con oroductos es:
120 0 0 0 0
```

Figure 10: Segunda versión mas de un cero

```
Ingrese el numero de elementos maximo 50
6
Ingrese el elemento 0
0
Ingrese el elemento 1
9
Ingrese el elemento 2
0
Ingrese el elemento 3
1
Ingrese el elemento 4
2
Ingrese el elemento 5
5

El arreglo original es:
0 9 0 1 2 5

El arreglo con oroductos es:
0 0 0 0 0 0
```

---



---

```

1  #include <stdio.h>
   #define ZERO 0
3  #define ONE 1
   #define SIZE 50
5
   void products(int [], int [], int lenght);
7  void print_array(int a[], int lenght);

9  void print_array(int ar[], int lenght)
   {
11     /*funcion que umprime el arreglo*/
       int i = 0;
13
       printf("\t\t");
15     while(i < lenght)
       {
17         printf("%d ", ar[i++]);
       }
19
       printf("\n\n");
21 }

23 void products(int a[], int product[], int lenght)
   {
25     int i = 0;

27     //el primer elemento de product es el
       //primer elemento de a
29     product[i] = a[i];
       i++;
31
       while(i < lenght)
33     {
           // el elemento anterior de prod
35         // por el elemento equi de a
           product[i] = product[i - ONE] * a[i];
37         i++;
       }
39
       printf("\t\tEl arreglo original es:\n");
41     print_array(a, lenght);
       printf(\
43     "\t\tEl arreglo con producto acumulado es:\n");
       print_array(product, lenght);
45 }

47 int main()
   {

```

```

49     int a[SIZE], tamaño, product[SIZE], i = 0;

51     printf("\t\tCuantos elementos tendra \
el arreglo (maximo 50)?\n");
53     scanf(" %d", &tamaño);

55     while(i < tamaño)
    {
57         printf("Agregue el elemento %d\n", i);
        scanf(" %d", &a[i++]);
59     }

61     products(a, product, i);

63     return ZERO;
}

```

---



---

Code 4: version2.c

---



---

```

#include <stdio.h>
2  #define SIZE 50
   #define ZERO 0
4  #define ONE 1

6  int read_array(int []);
   void print_array(int [], int);
8  int producto(int [], int, int);
   void products(int [], int [], \
10               int);

12 int read_array(int a[])
   {
14     int i = 0, number;

16     printf("\t\tIngrese el numero \
de elementos maximo 50\n\t\t");
18     scanf(" %d", &number);

20     while(i < number)
    {
22         printf("\t\tI\
ngrese el elemento %d\n\t\t", i);
24         scanf(" %d", &a[i++]);
    }

26

28     return number;
   }
30

   void print_array(int a[], int length)

```

```

32  {
34      int i = 0;
36      printf("\t\t");
37      while(i < length)
38      {
39          printf("%d ", a[i++]);
40      }
41      printf("\n\n");
42  }

44  int producto(int a[], int index, int length)
45  {
46      /*Realiza el producto de cada uno de los
47       elementos del arreglo*/
48
49      int i = ZERO, pro = ONE;
50
51      while(i < length)
52      {
53          if(i == index)
54          {
55              i++;
56              continue;
57          }
58
59          pro *= a[i++];
60      }
61
62      return pro;
63  }
64
65  void products(int a[], int product[],\
66               int length)
67  {
68      /*Asigna el producto del arreglo
69       a un elemento del nuevo arreglo
70       evitando el elemento i del
71       arreglo original*/
72
73      int i = ZERO, prod = ONE;
74      int count = ZERO, index;
75      int activate_index;
76
77      //cuantsa cuantos ceros
78      // hay en el arreglo
79      while(i < length)
80      {
91          if(a[i] == ZERO)

```

```

82         {
83             count++;
84             index = i;
85         }
86
87         i++;
88     }
89
90     switch(count)
91     {
92     case ZERO:// no hay ceros
93         activate_index = ZERO;
94         //menos uno es para poder
95         //multiplicar todos los elementos
96         prod = producto(a, -ONE, length);
97         break;
98     case ONE://hay un cero
99         // producto que tendra el elemento
100        //con cero
101        //ademas que se activa en el if
102        //si hay valor
103        activate_index = \
104        producto(a, index, length);
105        prod = ZERO;
106        break;
107    default:
108        activate_index = ZERO;
109        prod = ZERO;
110    }
111
112    i = ZERO;
113
114    while(i < length)
115    {
116        //existe solo un 0
117        if(activate_index)
118        {
119
120            if(i == index)
121            {
122                product[i] = activate_index;
123            }
124            else
125            {
126                product[i] = ZERO;
127            }
128        }
129        else
130        {

```

```

132         //existe mas de un 0
133         if(prod == ZERO)
134         {
135             product[i] = ZERO;
136         }
137         //no existe ningun 0
138         else
139         {
140             product[i] = prod/a[i];
141         }
142     }
143     i++;
144 }
145 }
146 }
147 }
148 }
149
150 int main()
151 {
152     int a[SIZE], b[SIZE], l_1;
153
154     //lee y obtiene #elementos
155     l_1 = read_array(a);
156     //obtiene arreglo productos
157     products(a, b, l_1);
158     printf("\n\t\tEl arreglo \
159 original es:\n");
160     print_array(a, l_1);
161     printf("\t\tEl arreglo con \
162 oroductos es:\n");
163     print_array(b, l_1);
164
165     return ZERO;
166 }

```

---

## Ejercicio 4

Escribe una función que reciba dos arreglos de enteros ordenados en forma no decreciente, y combine dichos arreglos en un solo arreglo también ordenado en forma no decreciente.

Firma de la función: `void merge(int a1[], int n1, int a2[], int n2, int result[], int n3)`

Se puede asumir que la longitud máxima de los arreglos `a1` y `a2` será 20, `n1` y `n2` representa el número de elementos en dichos arreglos.

## Respuesta

El algoritmo pide al usuario elementos de dos arreglos, estos datos deben de ser del tipo entero, podemos observar en la figura (11) y (12). como es que se muestra en pantalla la petición, es importante remarcar que el arreglo que se da debe de estar en un orden no decreciente, este concepto lo he tomado como que puede ser un arreglo desordenado o un arreglo ordenado de manera creciente, si esto no se cumple lanza un mensaje como el de la figura (15) y se termina el programa.

Una vez que se entregan los datos, el algoritmo procede a realizar el merge a los dos arreglos y después imprime en pantalla el nuevo arreglo. Por ultimo se le pregunta al usuario si quiere que el arreglo este ordenado estrictamente de manera creciente, si este oprime 1 en teclado el arreglo se ordena y después se imprime el resultado como en la figura (14). La manera en que se comprueba que un arreglo es verdaderamente no decreciente, es primero ordenando una copia del verdadero arreglo de manera decreciente, este después se compara con el original, y si es igual, entonces se rechaza figura (15). El mismo algoritmo de ordenamiento se utiliza al final cuando el usuario decide ordenar el arreglo de manera creciente. El algoritmo que se utiliza es "merge sort".

Figure 11: Primer arreglo

```
****Primer arreglo****
Agregue los elementos del arreglo
Cuantos elementos tendra el arreglo (maximo 20)?
3
Agregue el elemento 0
1
Agregue el elemento 1
5
Agregue el elemento 2
6
```

Figure 12: Segundo arreglo

```
****Segundo arreglo****
Agregue los elementos del arreglo
Cuantos elementos tendra el arreglo (maximo 20)?
4
Agregue el elemento 0
10
Agregue el elemento 1
3
Agregue el elemento 2
7
Agregue el elemento 3
8
```

Figure 13: Imprime/Merge

```
El arreglo a[] es
1 5 6
El arreglo b[] es
10 3 7 8
El arreglo merge es:
1 5 6 10 3 7 8
```

Figure 14: Opción ordenar

```
Desea ordenar en orden creciente?(si: 1, no: other)
1
El arreglo ordenado es
1 1 2 3 8 9 10
```

Figure 15: Entrada decreciente

```
****Primer arreglo****
Agregue los elementos del arreglo
Cuantos elementos tendra el arreglo (maximo 20)?
5
Agregue el elemento 0
5
Agregue el elemento 1
4
Agregue el elemento 2
3
Agregue el elemento 3
2
Agregue el elemento 4
1

Agregue un arreglo que no este en orden decreciente
```



---



---

```

1 //Giovanny Encinia
  //31/08/2021
3 #include <stdio.h>
  #include "functions.h"
5 #define ZERO 0
  #define ONE 1
7 #define SIZE 20

9 int main()
  {
11     int a[SIZE], b[SIZE], c[SIZE + SIZE];
        int a_c[SIZE], b_c[SIZE];
13     int tamaño_1, tamaño_2;
        char option;

15
        printf("\t\t****Primer arreglo****\n");
17     tamaño_1 = write_copia_comprueba(a, a_c);

19     if(!tamaño_1)
        return ZERO;

21
        printf("\t\t****Segundo arreglo****\n");
23     tamaño_2 = write_copia_comprueba(b, b_c);

25     if(!tamaño_2)
        return ZERO;

27
        printf("\t\tEl arreglo a[] es \n");
29     print_array(a, tamaño_1);

31     printf("\t\tEl arreglo b[] es \n");
        print_array(b, tamaño_2);
33

35     merge(a, tamaño_1, b,\
            tamaño_2, c, tamaño_1+tamaño_2);

37
        printf("\t\tEl arreglo merge es: \n");
39     print_array(c, tamaño_1 + tamaño_2);

41     printf("\t\tDesea ordenar en \
orden creciente?(si: 1, no: other)\n\t\t");
43     scanf("%c", &option);

45     if(option == '1')
        {
47         merge_sort(c, ZERO,\
            tamaño_1 + tamaño_2 - ONE, ONE);

```

```

49     printf("\t\tEl arreglo ordenado es\n");
    print_array(c, tamano_1+tamano_2);
51 }

53     return ZERO;
}

```

---

#### Code 6: functions.h

```

#ifndef FUNCTIONS
2  #define FUNCTIONS

4  void merge_sort(int [], int, int, int);
    void merge(int [], int, int [], \
6      int, int [], int);
    void merge_(int [], int, int, int, int);
8  void print_array(int [], int);
    int leer_arreglo(int []);
10 void copy_array(int [], int [], int);
    int write_copia_comprueba(int a[], int a_c []);
12
#endif // FUNCTIONS

```

---

#### Code 7: fuctions.c

```

1  //Giovanny Encinia
    //31/08/2021
3  #include <stdio.h>
    #define ZERO 0
5  #define ONE 1
    #define SIZE 20
7
    void merge(int a[], int l_1, int b[], \
9      int l_2, int c[], int l_3)
    {
11     /*Junta dos arreglos en un nuevo arreglo*/

13     int i = ZERO, j = ZERO, k = ZERO;

15     while(i < l_1)
    {
17         c[k] = a[i++];
        k++;
19     }

21     while(j < l_2)
    {
23         c[k] = b[j++];
        k++;

```

```

25     }

27 }

29 void merge_(int a[], int left, int mid, int right, int order)
30 {
31     /*Es el pado merge del algoritmo merge sort*/
32
33     int l_1, r_1, k, i = ZERO, j = ZERO;
34
35     l_1 = mid - left + ONE;
36     r_1 = right - mid;
37     //Arreglos que nos ayudan a almacenar
38     //divisiones del original
39     int L[l_1], R[r_1];
40
41     while(i < l_1)
42     {
43         //nuestro punto de referencia
44         //siempre es left, este cambia
45         //conforme avanza la recursividad
46         L[i] = a[left + i];
47         i++;
48     }
49
50     while(j < r_1)
51     {
52         R[j] = a[mid + j + ONE];
53         j++;
54     }
55
56     i = j = ZERO;
57     k = left;
58
59     while(i < l_1 && j < r_1)
60     {
61         //oder 1 creciente
62         // 0 decreciente
63         if(order?(L[i] < R[j]): (L[i] > R[j]))
64         {
65             a[k] = L[i++];
66         }
67         else
68         {
69             a[k] = R[j++];
70         }
71
72         k++;
73     }

```

```

75     //pueden haber sobrado datos
76     //sin ordenar, asi que se llena
77     //con el sobrente
78     while(i < l-1)
79     {
80         a[k] = L[i++];
81         k++;
82     }
83
84     while(j < r-1)
85     {
86         a[k] = R[j++];
87         k++;
88     }
89 }
90
91 void merge_sort(int a[], int left, int right, int order)
92 {
93     /*algoritmo merge sort*/
94
95     int mid;
96
97     if(left < right)
98     {
99         //buscamos el punto medio
100        mid = left + (right - left) / 2;
101
102        //recursividad a laprimera mitad
103        merge_sort(a, left, mid, order);
104        //segunda mitad
105        merge_sort(a, mid + ONE, right, order);
106        //realiza merge a las mitades
107        merge_(a, left, mid, right, order);
108    }
109 }
110
111 }
112
113 void print_array(int a[], int size)
114 {
115     int i = ZERO;
116
117     printf("\t\t");
118     while(i < size)
119     {
120         printf("%d ", a[i++]);
121     }
122
123     printf("\n");
124 }

```

```

125 int leer_arreglo(int array[])
127 {
128     /*Lee cada uno de los elementos
129     que tendra el arreglo
130     el tamaño máximo es de 20 elementos*/
131
132     int tamaño, i = ZERO;
133
134     printf("\t\tCuantos elementos tendrá \
135 el arreglo (máximo 20)?\n\t\t");
136     scanf("%d", &tamaño);
137
138     while(i < tamaño)
139     {
140         printf("\t\tAgregue el elemento %d\n\t\t", i);
141         scanf("%d", &array[i++]);
142     }
143
144     printf("\n\n");
145
146     return tamaño;
147 }
148
149 int compare(int a[], int b[], int tamaño)
150 {
151     /*Compara cada elemento de los arreglos
152     y regresa 1 si son iguales ambos arreglos
153     0 si son distintos*/
154
155     int i = ZERO;
156
157     while(i < tamaño)
158     {
159         if(a[i] != b[i])
160         {
161             return ZERO;
162         }
163
164         i++;
165     }
166
167     return ONE;
168 }
169
170 void copy_array(int a[], int copy[], int size)
171 {
172     /*Copia elementos a un nuevo arreglo*/
173
174     int i = ZERO;

```

```

175     while(i < size)
177     {
179         copy[i] = a[i];
181         i++;
182     }
183
184 int write_copia_comprueba(int a[], int a_c[])
185 {
186     int tamanio;
187
188     printf("\t\tAgregue los \
189 elementos del arreglo\n");
190     tamanio = leer_arreglo(a);
191     copy_array(a, a_c, tamanio);
192     merge_sort(a_c, ZERO, tamanio - ONE, ZERO);
193
194     // comprueba que no este en orden decreciente
195     if(compare(a, a_c, tamanio))
196     {
197         printf("\t\tAgregue un arreglo \
198 que no este en orden decreciente\n");
199
200         return ZERO;
201     }
202
203     return tamanio;
204 }

```

---



---