

SISTEMAS INFORMÁTICOS

GITT, GIT, GIST Y GIEC

PRÁCTICA 3: Arrays Sentencias de control

UAH, Departamento de Automática, ATC-SOL

OBJETIVOS

- Comprender y manejar sentencias de control de flujo.
- Realizar un programa dado el pseudocódigo.
- Manejo de arrays.
- Utilizar métodos de ordenación y de búsqueda en arrays.

TEMPORIZACIÓN

- Inicio de la práctica: Semana del 16 de octubre
- Tiempo de desarrollo de la práctica: 3 semanas. Finalizarla antes del examen parcial 2
- Evaluación: ver fechas en el aula virtual

1. Introducción

El pseudocódigo (falso lenguaje) es comúnmente utilizado por los programadores para omitir secciones de código o para dar una explicación del paradigma que tomó el mismo programador para hacer sus códigos. Esto quiere decir que el pseudocódigo no es programable pero facilita la comprensión del problema que se pretende programar.

El principal objetivo del pseudocódigo es el de representar la solución a un algoritmo de la forma más detallada posible, y a su vez lo más parecida posible al lenguaje que posteriormente se utilizará para la codificación del mismo.

A continuación, se plantea la realización de varios programas empleando conocimientos básicos de tipos de datos y sentencias de control. En algunos de ellos, se le propone un pseudocódigo con el algoritmo que el programa debe realizar. Recuerde que no es código fuente en lenguaje C, sino simplemente una descripción informal de lo que debe hacer el código del programa.

2. Ejercicios a realizar

Las actividades a realizar en esta práctica están divididas en dos apartados: Sentencias de Control y Arrays. Cada apartado se centra en un tema pero tienen como un objetivo marcado el trabajar con pseudocódigo que pueda ser traducido a un lenguaje de programación concreto, en este caso C. Lea detenidamente cada apartado y complete las actividades.

2.1 Sentencias de control

En general, la ejecución secuencial es el orden en el que se procesan las sentencias de un programa, así una sentencia es ejecutada después que la anterior y después la siguiente, la siguiente ... pero varias sentencias en C rompen con este orden secuencial, son las llamadas Sentencias de Control. Estas Sentencias de Control se pueden clasificar en dos grupos: selectivas (if, if ... else, switch) y repetitivas (for, while, do .. while). En el siguiente apartado se tienen que usar estas Sentencias, selectivas y repetitivas, para implementar dos funciones: Potencia() y Factorial(). Cada una de las funciones van acompañada de un pseudocódigo que tendrá que ser codificado en C.

2.1.1 Ejemplo de pseudocódigo.

A continuación, se muestra en la Figura- 1 el pseudocódigo para realizar un sencillo menú con 3 opciones, seguido de la codificación en C de dicho pseudocódigo.

En este apartado no debe de escribir ningún código sólo de estudiar y comprobar la relación existente entre el código en C mostrado y el pseudocódigo utilizado para generarlo.

```
Declarar variables
Entero Opcion = 0
MIENTRAS Opcion < 3
    Mostrar "Elija una opción:
        1 – Calcular la potencia
        2 – Calcular el factorial
        3 – Salir
    Solicitar Opcion
    SI Opcion = 1
        Leer de teclado los datos necesarios para el cálculo y llamar a la
        función POTENCIA
    SI Opcion = 2
        Leer de teclado los datos necesarios para el cálculo y llamar a la
        función FACTORIAL
    FINSI
Repetir
```

Figura- 1. Menú

Codificación en C

```
#include <stdio.h>

int potencia (int bas, int exp);
int factorial(int num);

int main ()
{
    int opcion = 0, base, exponent, numero;
    while (opcion < 3)
    {
        printf ("Elija una opción:\n ");
        printf ("1 – Calcular la potencia ");
        printf ("2 – Calcular número del factorial ");
        printf ("3 – Salir ");

        scanf ("%d",&opcion);
        if (opcion == 1)
```

```
{
    printf("Introduzca la base:");
    scanf("%d", &base);
    do
    {
        printf("Introduzca el exponente:");
        scanf("%d", &exponente);
    }while(exponente < 0);

    printf ("El resultado es:%d\n", potencia(base,
                                                exponente));
}
if (opcion == 2)
{
    printf("Introduzca un numero:");
    scanf("%d", &numero);
    printf ("El factorial es:%d\n", factorial(numero);
}

} //FIN while

return 0;
} //FIN main

int potencia (int base, int exponente)
{
    // Hay que implementarla en el apartado siguiente
}
int factorial (int numero)
{
    // Hay que implementarla en el apartado siguiente
}
```

2.1.2 Cálculo de la potencia de un número

En este apartado va a trabajar con la función que calcula la potencia de dos números pedidos por teclado. Para ello utilice la función `potencia()` proporcionada más adelante y utilice el código de la función `main()`, del apartado anterior 2.1.1, para completarlo y escribir la definición de la función.

La función `main()` realiza las siguientes operaciones:

- Pedir al usuario por teclado dos valores (base y exponente).
- Comprobar que el valor exponente es correcto (que no sea negativo).
- Utilizar la función `potencia()` proporcionada.
- Mostrar por pantalla el resultado calculado.

A continuación, se muestra en la Figura- 2 el pseudocódigo de una función que recibe dos valores, calcula y devuelve el resultado de elevar el primer valor al segundo (potencia de un número). Así mismo más abajo está la codificación en C de dicho pseudocódigo.

Función Potencia (recibe dos enteros Base y Exponente)

```
Declarar entero Resultado = 0
Resultado = Base

SI (Exponente == 0) Entonces
    Resultado = 1
SI NO
    MIENTRAS (Exponente > 1)
        Resultado = Resultado * Base
        Exponente = Exponente - 1
    Fin Mientras
FIN SI

Devolver Resultado
```

Figura- 2. Algoritmo para obtener la N-ésima potencia de un número dado

Codificación en C

```
int potencia (int base, int exponente)
{
    int resultado = 0;

    resultado = base;

    if (exponente == 0)
        resultado = 1;
    else
        while (exponente > 1)
        {
            resultado*=base;
            exponente--;
        }

    return resultado;
}
```

2.1.3 Cálculo del factorial de un número

Codifique en una función el algoritmo para calcular el factorial de un número usando el pseudocódigo que aparece en la Figura- 3.

Para comprobar su funcionamiento, se usa el mismo programa del apartado 2.1.1, complete dicho código con la definición de la función `factorial`. La opción referida a la función factorial debe pedir al usuario el número sobre el que calculará el factorial, debe de comprobar que dicho número sea correcto (mayor o igual que cero), esto debe ser añadido en el código de la función `main` proporcionada en el apartado 2.1.1, y llama a la función implementada mostrando por pantalla el resultado devuelto por la función.

```
Función Factorial (recibe un entero Numero)
    Declarar entero Resultado

    SI Numero == 0
        Resultado = 1
    SI NO
        Resultado = Numero
    FIN SI

    MIENTRAS (Numero > 1)
        Numero = Numero - 1
        Resultado = Resultado * Numero
    FIN MIENTRAS

    Devolver Resultado
```

Figura- 3. Factorial de un número

Modifique las funciones `potencia` y `factorial` para que utilicen una sentencia `for` en lugar de una sentencia `while`.

2.2 Arrays

Dentro del conjunto de elementos de almacenamiento de información con los que se puede trabajar en C se encuentran los arrays. Un array es una colección de variables del mismo tipo referenciadas por un nombre común seguido de un índice que indica la posición del elemento dentro del array. Los elementos de un array se almacenan en posiciones contiguas de memoria. La dirección más baja corresponde al primer elemento y la más alta al último. En un array de N elementos, el índice del primero es 0 y el índice del último es N - 1.

El siguiente programa define un array de diez elementos de tipo entero, almacena en él diez números leídos desde teclado y finalmente visualiza el contenido del array por pantalla, es decir, el valor de cada uno de sus elementos:

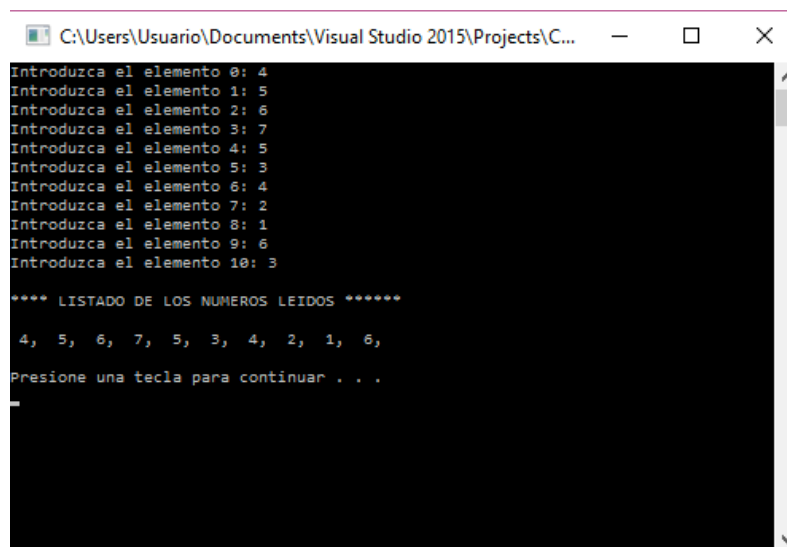
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX 10
4
5 int main ()
6 {
7     int numeros[MAX], i;
8
9     for (i=0; i<MAX; i++)
10    {
11        printf("Introduzca el elemento %d: ", i);
12        scanf(" %d" ,&numeros[i]);
13    }
14    printf("\n**** LISTADO DE LOS NUMEROS LEIDOS ****\n\n");
15
```

```
16   for (i=0; i<MAX; i++)
17       printf(" %d, ", numeros[i]);
18
19   printf("\n\n");
20
21   return 0;
22 }
```

Al trabajar con los arrays hay que tener cuidado con no exceder sus límites, puesto que si se ha definido el array `numeros` con 10 elementos el elemento undécimo ya no pertenece a este array, ese elemento se encuentra fuera del conjunto de elementos de `numeros`. Modifique la línea 9 conforme a la siguiente expresión:

```
for (i = 0; i<=MAX; i++)
```

Vuelva a compilar el programa y ejecútelo para comprobar qué sucede al incluir esa modificación. En la Figura- 4 se puede observar la salida del programa habiendo incluido la entrada y salida de datos del elemento extra del array `numeros`. En la Figura- 5 se puede ver el mensaje que aparece una vez que el programa finaliza su ejecución. En ese mensaje se da a conocer que la variable `numeros` se ha corrompido. El motivo de esta corrupción es que se ha usado un espacio de memoria para el que el array no tiene permisos.



```
C:\Users\Usuario\Documents\Visual Studio 2015\Projects\C...
Introduzca el elemento 0: 4
Introduzca el elemento 1: 5
Introduzca el elemento 2: 6
Introduzca el elemento 3: 7
Introduzca el elemento 4: 5
Introduzca el elemento 5: 3
Introduzca el elemento 6: 4
Introduzca el elemento 7: 2
Introduzca el elemento 8: 1
Introduzca el elemento 9: 6
Introduzca el elemento 10: 3

**** LISTADO DE LOS NUMEROS LEIDOS ****

4, 5, 6, 7, 5, 3, 4, 2, 1, 6,
Presione una tecla para continuar . . .
```

Figura- 4. Consola de ejecución el programa

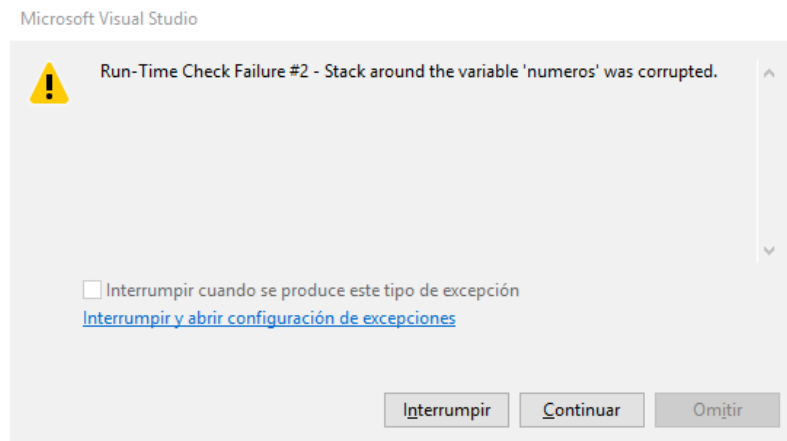


Figura- 5. Cuadro de advertencia de un error

Recupere la línea 9 original en el código fuente:

```
for (i = 0; i<MAX; i++)
```

y modifique ahora la línea 16 con la expresión:

```
for (i = 0; i<=MAX; i++)
```

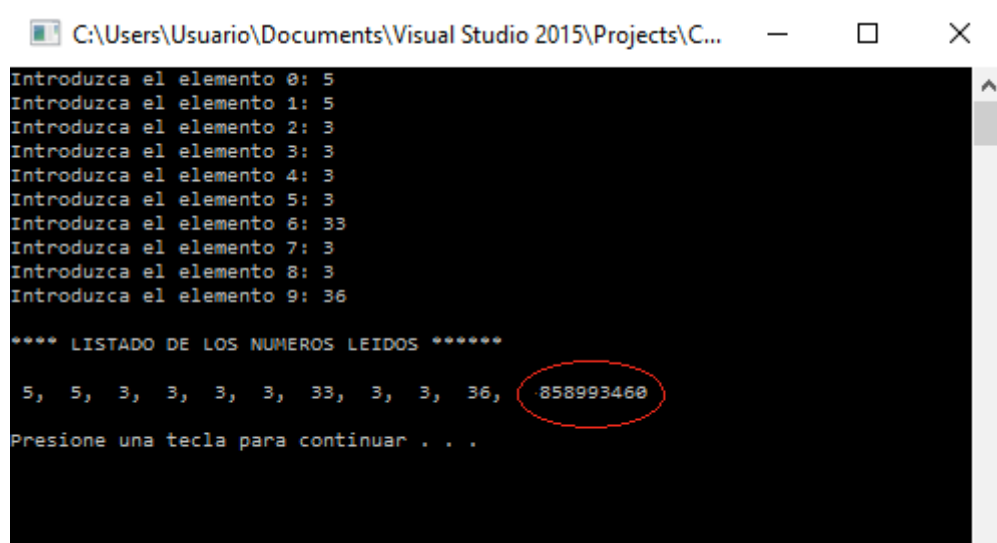


Figura- 6. Resultado de la ejecución del programa

Razone por qué aparece el elemento undécimo en el “LISTADO DE NUMEROS LEIDOS” como negativo.

En los siguientes apartados se proponen una serie de programas a desarrollar que usarán el array como elemento principal de almacenamiento.

2.2.1 Desglose de euros en billetes y monedas

En este apartado se debe completar un programa que, dada una cantidad de euros sin céntimos, la desglose en el menor número posible de billetes y monedas.

Por ejemplo:

- 1747 euros se desglosan en,
- 3 billetes de 500 euros
 - 1 billete de 200 euros
 - 2 billetes de 20 euros
 - 1 billete de 5 euros
 - 1 moneda de 2 euros

En el programa a completar se usará un array llamado `valor`, con nueve elementos de tipo entero, para almacenar en orden descendente el valor de los distintos billetes y monedas de valor mayor o igual a 1 euro (billetes de 500 euros, de 200 euros, de 100 euros, de 50 euros, de 20 euros, de 10 euros y de 5 euros; monedas de 2 euros y de 1 euro).

La Figura- 7 muestra el pseudocódigo de la operación a realizar para calcular dicho desglose del valor almacenado en la variable `cantidad` (es decir, este pseudocódigo parte de que la variable `cantidad` ya tiene asignado el valor a desglosar):

```
i = 0
MIENTRAS (cantidad > 0)
    n = cantidad / valor[i] (división entera)
    cantidad = resto de la división anterior
    SI (n > 0)
        Mostrar "n billetes/monedas de valor[i] euros"
    Incrementar i
```

Figura- 7. Pseudocódigo para el desglose de monedas.

Codificación en C

Completar el siguiente código mediante la codificación de la función cuya declaración se muestra a continuación y que realiza el desglose de una cantidad en una serie de valores según el pseudocódigo mostrado en la Figura- 7:

```
void cuenta_billetes_monedas(int cantidad, int valor[]);
```

Observar que la función `cuenta_billetes_monedas()` no pide datos al usuario por teclado, los recibe como parámetros desde la función `main()`, pero sí muestra el resultado por pantalla sin devolver ninguna información al dicha función `main()`.

Código a completar:

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

void cuenta_billetes_monedas(int, int[]);

int main ()
{
    int valor[9] = { 500, 200, 100, 50, 20, 10, 5, 2, 1 };
    int cantidad;

    printf ("Practica 3, ejercicio 2\n");

    printf ("Introduce una cantidad de euros (sin centimos): ");
    scanf ("%d", &cantidad);

    if (cantidad<=0)
    {
        printf ("ERROR: La cantidad debe ser mayor que cero.\n");
        system ("pause");
        return -1;
    }

    cuenta_billetes_monedas(cantidad, valor);
    system ("pause");
    return 0;
}

//Completar aquí la definición de la función a realizar
```

2.2.2 Ordenación mediante el algoritmo de inserción

El algoritmo de inserción es el que con más frecuencia usa el ser humano para ordenar información. Este algoritmo consiste en dividir el total de números a ordenar en dos grupos: el grupo de los números casi-ordenados y el grupo de los números desordenados. Para poder explicar mejor el algoritmo se va a usar un caso práctico, de tal forma que la lista inicial de números a ordenar es la siguiente:

Ejemplo: 34 32 23 25

En el momento de iniciar la ordenación todos los números se encuentran en el conjunto de los desordenados, así el primero más a la izquierda de la lista pasa a ser el único integrante de los casi-ordenados.

Casi-ordenados
34

Desordenados
32 23 25

A continuación, se procede con la primera iteración donde el siguiente número de la lista de los números desordenados, 32, se ordena respecto a los elementos de la lista de los

casi-ordenados, buscando su posición correcta mediante comparaciones de derecha a izquierda de la lista, en esta ocasión al existir un único elemento en el grupo de los casi-ordenados sólo será necesaria una comparación para encontrar el lugar correcto, ¿es 32 menor que 34?

Casi-ordenados
32 34

Desordenados
23 25

En la siguiente iteración será el número 23 el que se compare con el grupo de los casi-ordenados para ubicarlo correctamente. Primero se compara con 34 y ya que el 23 es menor continuamos con la siguiente comparación. Se compara también con 32 y en esta ocasión 23 continúa siendo menor. Como ya no quedan más números en la lista se finaliza ubicando el 23 en la posición más a la izquierda como número menor.

Casi-ordenados
23 32 34

Desordenados
25

Llegando a este punto la ordenación está casi completada, el único número que queda en la lista de desordenados es el 25 así que se procede a posicionarlo en la lista de casi-ordenados en lugar correcto. El número de comparaciones a realizar es de tres: ¿es 25 menor que 34? ¿es 25 menor que 32? ¿es 25 menor que 23? En la última comparación el resultado es no por lo que el número 25 se debe de ubicar justo detrás del 23.

Casi-ordenados
23 25 32 34

Desordenados

Esta sería la última iteración del algoritmo ya que el grupo de los números desordenados se ha reducido a cero. El grupo de casi-ordenados pasa a ser el grupo de los números ordenados.

Para poder codificar este algoritmo en C es necesario contar con varios elementos. En primer lugar, un array que almacene la lista completa de números a ordenar. En segundo lugar, un bucle que recorra los números desordenados de izquierda a derecha, y en este bucle una variable que permita almacenar el número que se debe ordenar a continuación, es decir, el número más a la izquierda del grupo de desordenados. En tercer lugar, otro bucle, anidado al anterior que permita recorrer el grupo de números casi-ordenados de derecha a izquierda, y así poder realizar las comparaciones.

En la explicación del algoritmo se han usado dos grupos de elementos, casi-ordenados y desordenados. En la implementación del algoritmo los elementos de los dos grupos se encuentran incluidos en un solo array, y se diferencia un grupo de otro por el índice de acceso, para el grupo de elementos desordenados se usará el índice *i*, cuyo valor lo actualiza el bucle exterior, y para el grupo de los casi-ordenados se usa el índice *j*, actualizado por el bucle anidado al exterior, ver Figura- 8 para diferenciar los dos bucles e índices.

Teniendo en cuenta todo lo anterior realice un programa que, dada una serie de N números (100 números como máximo) de tipo double (coma flotante de doble precisión), los ordene de menor a mayor mediante el algoritmo de inserción.

La Figura- 8 muestra una propuesta para el pseudocódigo.

```
Preguntar cuántos números (n) quiere introducir el usuario
Pedir los n números y guardarlos en un array
Presentar los números por pantalla
Para un índice i=1 hasta n
    Índice auxiliar j igual i-1
    Variable temporal igual al valor de elemento i del array.
    Mientras valor elemento j de array > variable temporal y el índice j mayor o igual que cero
        Valor array j+1 igual a valor array j
        Decrementar valor índice j
    Valor del array j+1 igual variable temporal.
Imprimir el array ordenado
```

Figura- 8. Pseudocódigo para el algoritmo de inserción.

2.2.3 Búsqueda binaria

Realice un programa que implemente el método de búsqueda binaria (descrito a continuación). Para aplicarlo será necesario disponer de un array de enteros ordenados de menor a mayor (100 números como máximo). Se puede optar por pedir los números y entonces ordenarlos, o por exigir (y verificar) que el usuario los introduzca ya ordenados.

La búsqueda binaria se apoya en el hecho de que los datos están ordenados, para localizar el elemento buscado en muy pocos pasos. Se busca en un intervalo del array (inicialmente todo el array) y se va reduciendo ese intervalo a la mitad en cada paso. Para ello, basta comparar el elemento buscado con el elemento situado en el centro del intervalo de búsqueda. Si el elemento buscado es mayor, hay que continuar la búsqueda por la mitad superior (la inferior queda descartada junto con el elemento comparado). Si es menor, hay que continuar por la mitad inferior. Si es igual, ya se ha encontrado el elemento buscado. Si el intervalo se consume completamente sin que se haya encontrado, entonces el elemento no está en el array¹.

La Figura- 9 muestra una propuesta para el pseudocódigo.

```
Preguntar cuántos números (n) quiere introducir el usuario
Pedir los n números y guardarlos en un array
Ordenarlos, o verificar que el usuario los ha introducido ordenados
Pedir el valor (x) a buscar
bajo = 0
alto = n-1
MIENTRAS (bajo <= alto)
    central = (alto + bajo) / 2 (división entera)
    SI (x == número[central])
        ¡Éxito! Mostrar al usuario la posición central
        Salir del programa
    SI (x > número[central])
        bajo = central + 1
    SI (x < número[central])
        alto = central - 1
¡Fracaso! El número buscado no estaba en el array
```

Figura- 9. Algoritmo de búsqueda binaria.

2.2.4 Menú

El siguiente ejercicio consta de dos partes:

2.2.4.1 Primera parte: codificación del programa

Realice un programa que genere un menú con las siguientes opciones:

1. **Introducir array**
2. **Visualizar array**
3. **Ordenar array**
4. **Buscar un elemento en el array**
5. **Modificar un elemento del array**
6. **Salir**

Las opciones del menú se realizarán sobre un array de 10 elementos de tipo `int` definido como una variable local en la función `main()`. El usuario introducirá una opción entre 1 y 6. El programa debe asegurar que la opción introducida está entre estos límites:

1. Si la opción introducida no se encuentra entre 1 y 6 el programa mostrará de nuevo el menú para que el usuario introduzca una de las opciones que se le muestran.
2. Si el usuario pulsa una opción entre 1 y 5, el programa ejecutará la opción descrita en la opción pulsada y volverá de nuevo al menú para repetir el proceso.
3. Si el usuario pulsa la opción 6 (*Salir*), el programa terminará.
4. Para ejecutar las distintas opciones del menú deberá realizar, además de la función `main()`, las siguientes funciones:
 - `int Menu()`: función que visualiza el menú, lee y retorna la opción pulsada por el usuario.
 - `int introducir_array(int Array[])`: función que lee los elementos del array desde teclado y retorna el número de elementos introducidos. La función debe asegurar que el número de elementos está entre 0 y 10.
 - `void visualizar_array(int Array[], int n_eltos)`: función que visualiza los elementos del array.
 - `void ordenar_insercion (int A[], int n_eltos)`: función que ordena los elementos del array. Utilice la función que a tal efecto programó en el ejercicio 2.2.2.
 - `int busqueda_binaria(int x, int valores[], int n_eltos)`: función que busca el elemento `x` en el array. Retorna la posición del elemento en el array o -1 si no se encontró. Utilice la función que a tal efecto programó en el ejercicio 2.2.3.
 - `int introducir_valor(int x, int Array[], int pos)`: función que almacena el valor `x` en la posición `pos` del array. Retorna 1 si el valor se añade correctamente o -1 si la posición supera el límite del array.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
#define SALIR 6

int Menu();
int introducir_array(int []);
void visualizar_array(int Array[], int n_eltos);
void ordenar_insercion (int A[], int num);
int busqueda_binaria(int x, int valores[], int num);
int introducir_valor(int, int [], int);

int main()
{
    int Array[MAX], n_eltos=0;
    int op=0, x, pos, resu;
    while ((op=Menu()) != SALIR)
```

```
{
    switch(op)
    {
        case 1:
            n_eltos = introducir_array(Array);
            break;

        case 2:
            visualizar_array(Array, n_eltos);
            system("pause");
            break;

        case 3:
            ordenar_insercion(Array, n_eltos);
            break;

        case 4:
            Introducir el valor a buscar y leerlo
            pos = busqueda_binaria(x, Array, n_eltos);
            if (pos == -1)
                Imprimir en pantalla que es un fracaso
            else
                Imprimir en pantalla el valor de la posición del
                valor buscado
                system("pause");
                break;

        case 5:

            1. Lectura desde teclado del elemento a buscar en el
                array.
            2. Llamada a la función busqueda_binaria(), que le
                permitirá encontrar la posición que ocupa el
                elemento en el array.
            3. Si el elemento NO se encuentra en el array se
                imprimirá un mensaje y se dará por finalizada la
                operación. Si el elemento SI se encuentra en el
                array, se leerá desde teclado el nuevo valor a
                introducir y se realizará una llamada a la
                función introducir_valor(), que le permitirá
                realizar la modificación del elemento
                system("pause");
                break;

        case 6:
            printf ("Saliendo del programa.....\n\n ");
    }
}

int Menu()
{
    int op;
    system("cls");
    do
    {
        system("cls");
```

```
printf("\n");  
printf("\t1. Introducir array\n");  
printf("\t2. Visualizar array\n");  
printf("\t3. Ordenar array\n");  
printf("\t4. Buscar un elemento en el array\n");  
printf("\t5. Modificar un elemento del array\n");  
printf("\t6. Salir\n");  
printf("\n Selecciones la opcion deseada\n\n");  
scanf("%d",&op);  
}while (op < 1 || op > SALIR);  
return op  
}
```

Funciones a realizar

2.2.4.2 Segunda parte: organización del programa en distintos archivos

Una vez terminado y ejecutado correctamente el programa, organice el código en los siguientes archivos:

- **Menu.h**: debe contener las directrices al preprocesador situadas al principio del programa, antes del comienzo de la función `main()`.
- **Menu.c**: debe contener la función `main()`. Con el fin de incluir las directrices al preprocesador, la función `main()` debe ir precedida de la siguiente línea de código:

```
#include "Menu.h"
```

- **Funciones.c**: debe contener el código de todas las funciones utilizadas. Al igual que `main()` debe ir precedidas por la línea de código:

```
#include "Menu.h"
```

Los tres archivos deben ser añadidos al proyecto.