

Laboratorio 1
Estructuras de Datos

Santiago Cardona 160004910
Juan Aristizabal 160004903

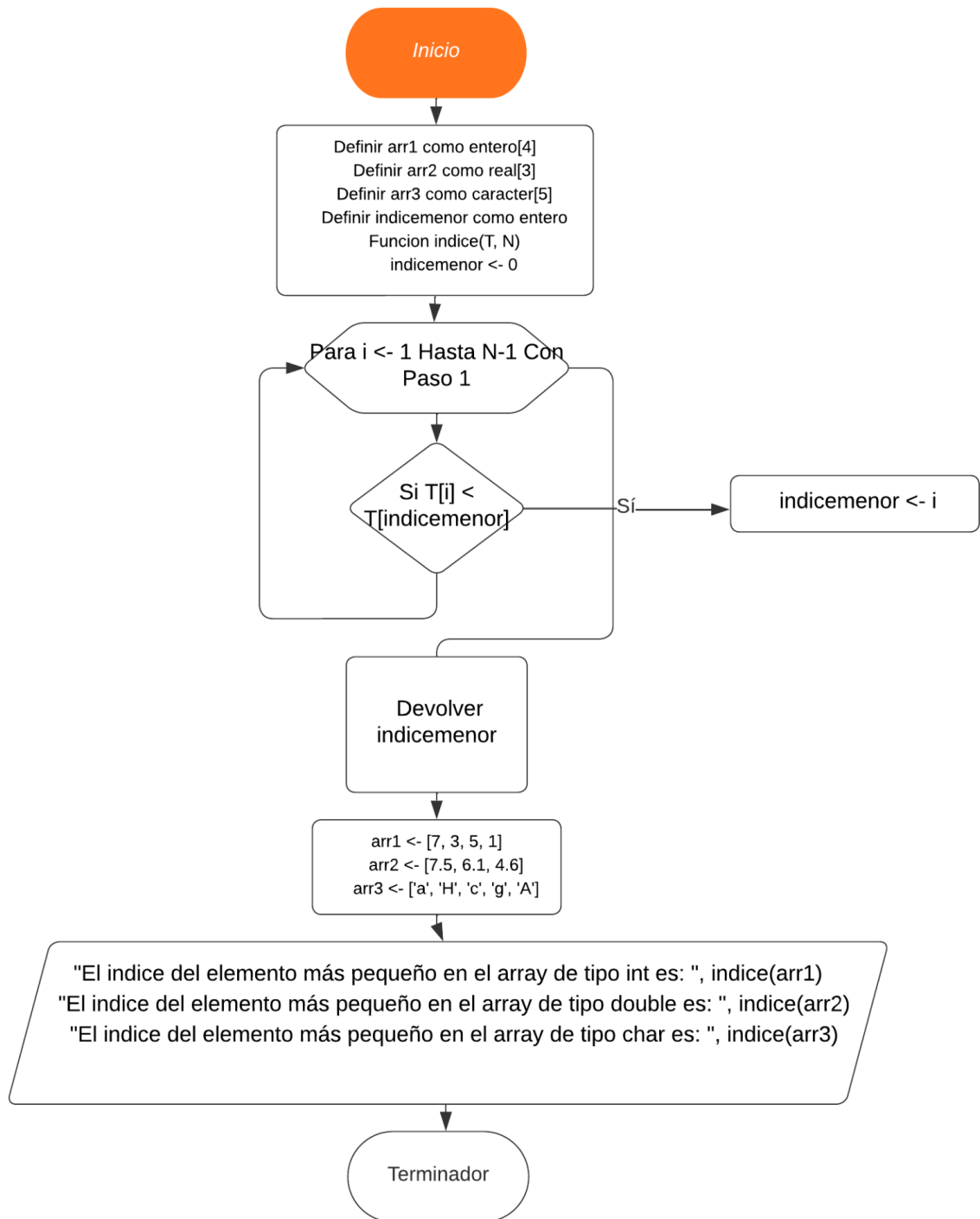
Agosto 31, 2023

1. Punto 1

1.1 Enunciado

Escriba una función template para encontrar el índice del elemento más pequeño de un array de cualquier tipo. Pruebe la función con tres arrays de tipo int, double y char. Entonces imprima el valor del elemento más pequeño.

1.2 Diagrama de flujo



1.3 Pseudocódigo

Algoritmo Ejercicio_1

Definir arr1 como entero[4]

Definir arr2 como real[3]

Definir arr3 como caracter[5]

Definir indicemenor como entero

Funcion indice(T, N)

```

    indicemenor <- 0
    Para i <- 1 Hasta N-1 Con Paso 1 Hacer
        Si T[i] < T[indicemenor] Entonces
            indicemenor <- i
        Fin Si
    Fin Para
    Devolver indicemenor
FinFuncion

```

```

arr1 <- [7, 3, 5, 1]
arr2 <- [7.5, 6.1, 4.6]
arr3 <- ['a', 'H', 'c', 'g', 'A']

```

```

    Escribir("El indice del elemento más pequeño en el array de tipo int es: ",
    indice(arr1))

```

```

    Escribir("El indice del elemento más pequeño en el array de tipo double es: ",
    indice(arr2))

```

```

    Escribir("El indice del elemento más pequeño en el array de tipo char es: ",
    indice(arr3))

```

```

FinAlgoritmo

```

1.4 Código fuente

```

template<typename T, int N>
int indice(&array)[N]{
    int indicemenor = 0;
    for (int i = 1; i < N; i++) {
        if (array[i] < array[indicemenor]) {
            indicemenor = i;
        }
    }
    return indicemenor;
}

int main() {
    int arr1[4] = {7, 3, 5, 1};
    double arr2[3] = {7.5, 6.1, 4.6};
    char arr3[5] = {'a', 'H', 'c', 'g', 'A'};

    cout << "El indice del elemento más pequeño en el array de
    tipo int es: " << indice(arr1) << endl;
    cout << "El indice del elemento más pequeño en el array de
    tipo double es: " << indice(arr2) << endl;
    cout << "El indice del elemento más pequeño en el array de
    tipo char es: " << indice(arr3) << endl;

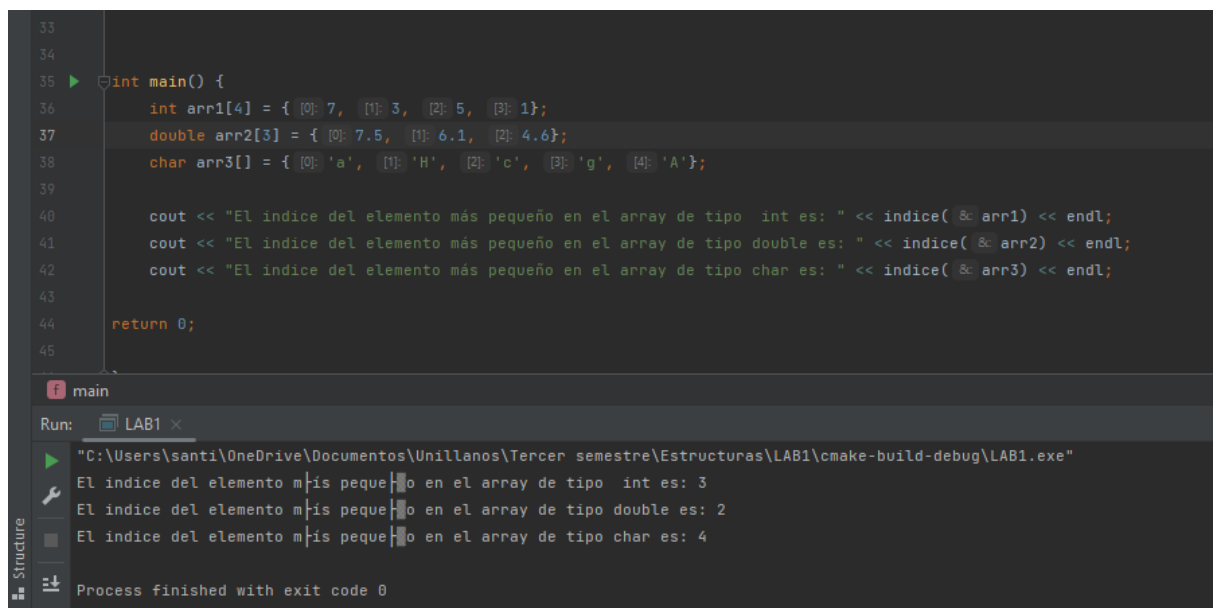
    return 0;
}

```

1.5 Descripción

La función índice toma como entrada un array y devuelve el índice del elemento más pequeño en el array. En el main se crean tres arrays de diferentes tipos de datos. El primer array es de tipo int, el segundo es de tipo double y el tercero es de tipo char. El código utiliza un bucle for para iterar a través del array y encontrar el índice del elemento más pequeño. Si se encuentra un elemento más pequeño, se actualiza la variable índicemenor con el valor de i (la posición más pequeña hasta el momento). Finalmente, la función devuelve el índice del elemento más pequeño según el tipo de dato, la función índice para cada uno de ellos como recibe una plantilla template como parámetro. Luego, imprime la posición del índice.

1.6 Resultados



```
33
34
35 int main() {
36     int arr1[4] = { [0]: 7, [1]: 3, [2]: 5, [3]: 1};
37     double arr2[3] = { [0]: 7.5, [1]: 6.1, [2]: 4.6};
38     char arr3[] = { [0]: 'a', [1]: 'H', [2]: 'c', [3]: 'g', [4]: 'A'};
39
40     cout << "El indice del elemento más pequeño en el array de tipo int es: " << indice( & arr1) << endl;
41     cout << "El indice del elemento más pequeño en el array de tipo double es: " << indice( & arr2) << endl;
42     cout << "El indice del elemento más pequeño en el array de tipo char es: " << indice( & arr3) << endl;
43
44     return 0;
45 }
```

main

Run: LAB1 x

"C:\Users\santi\OneDrive\Documentos\Unillanos\Tercer semestre\Estructuras\LAB1\cmake-build-debug\LAB1.exe"

El indice del elemento más pequeño en el array de tipo int es: 3

El indice del elemento más pequeño en el array de tipo double es: 2

El indice del elemento más pequeño en el array de tipo char es: 4

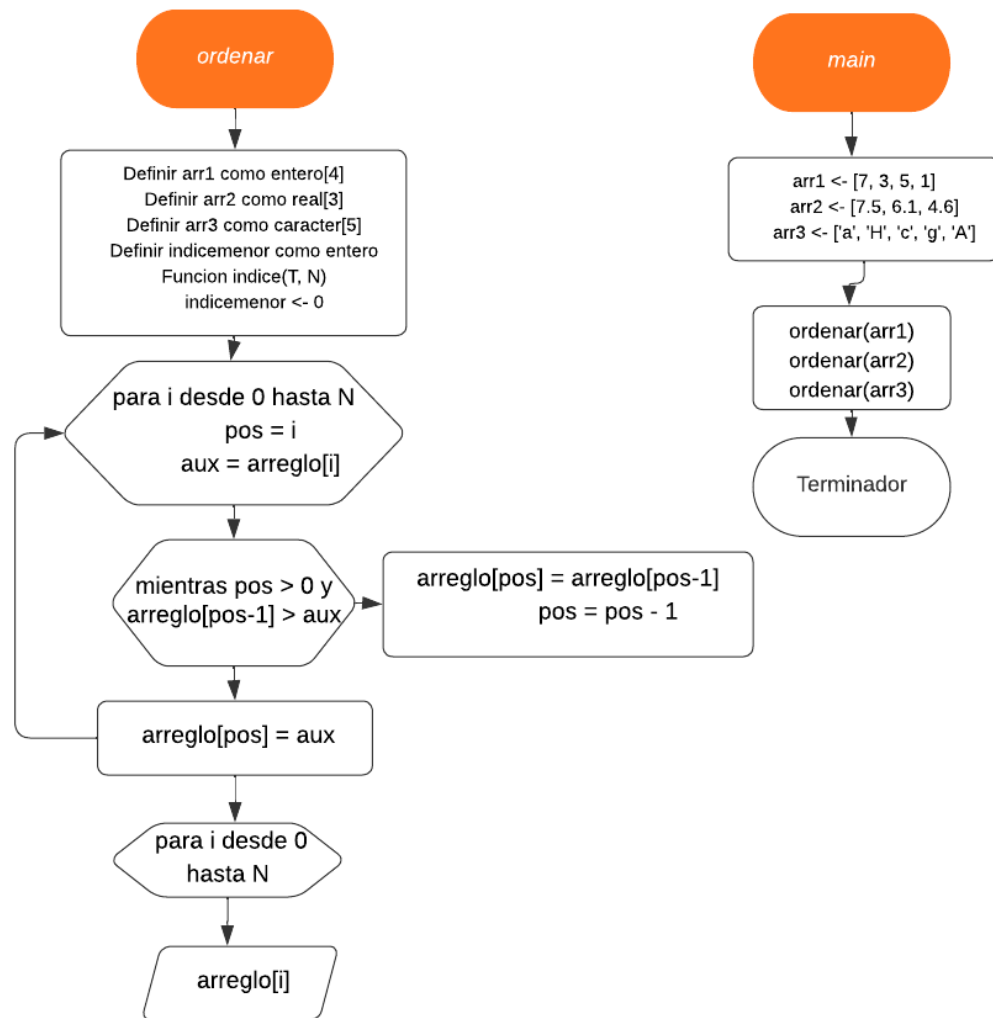
Process finished with exit code 0

2. Punto 2

2.1 Enunciado

Si encontramos el elemento más pequeño en un array, podemos ordenar un array usando el algoritmo selection sort. En este algoritmo, encontramos el elemento más pequeño en el array y lo intercambiamos con el primer elemento. Después encontramos el elemento más pequeño de los elementos restantes y lo intercambiamos con el segundo elemento. Continuamos hasta que el array esté completamente ordenado. Escriba un programa que ordene tres arrays de tipo int, double y char.

2.2 Diagrama de flujo



2.3 Pseudocódigo

```

función ordenar(arreglo)
  para i desde 0 hasta N
    pos = i
    aux = arreglo[i]
    mientras pos > 0 y arreglo[pos-1] > aux
      arreglo[pos] = arreglo[pos-1]
      pos = pos - 1
    fin mientras
    arreglo[pos] = aux
  fin para
  para i desde 0 hasta N
    imprimir arreglo[i]
  fin para
fin función
  
```

```

arr1 = {7, 3, 5, 1}
arr2 = {7.5, 6.1, 4.6}
  
```

```
arr3 = {'a', 'H', 'c', 'g', 'A'}
```

```
ordenar(arr1)
```

```
ordenar(arr2)
```

```
ordenar(arr3)
```

2.4 Código fuente

```
template<typename T, int N>
void sort(T(&array) [N]){
//ordenamiento por inserción
    T aux;
    for(int i=0; i < N; i++){
        int pos = i;
        aux = array[i];
        while((pos>0) and (array[pos-1]>aux)){
            array[pos] = array[pos-1];
            pos--;
        }
        array[pos] = aux;
    }
    for(int i=0;i<N;i++){
        cout<<array[i]<<" ";
    }
    cout<<endl;
}

int main() {
    int arr1[4] = {7, 3, 5, 1};
    double arr2[3] = {7.5, 6.1, 4.6};
    char arr3[] = {'a', 'H', 'c', 'g', 'A'};

    sort(arr1);
    sort(arr2);
    sort(arr3);

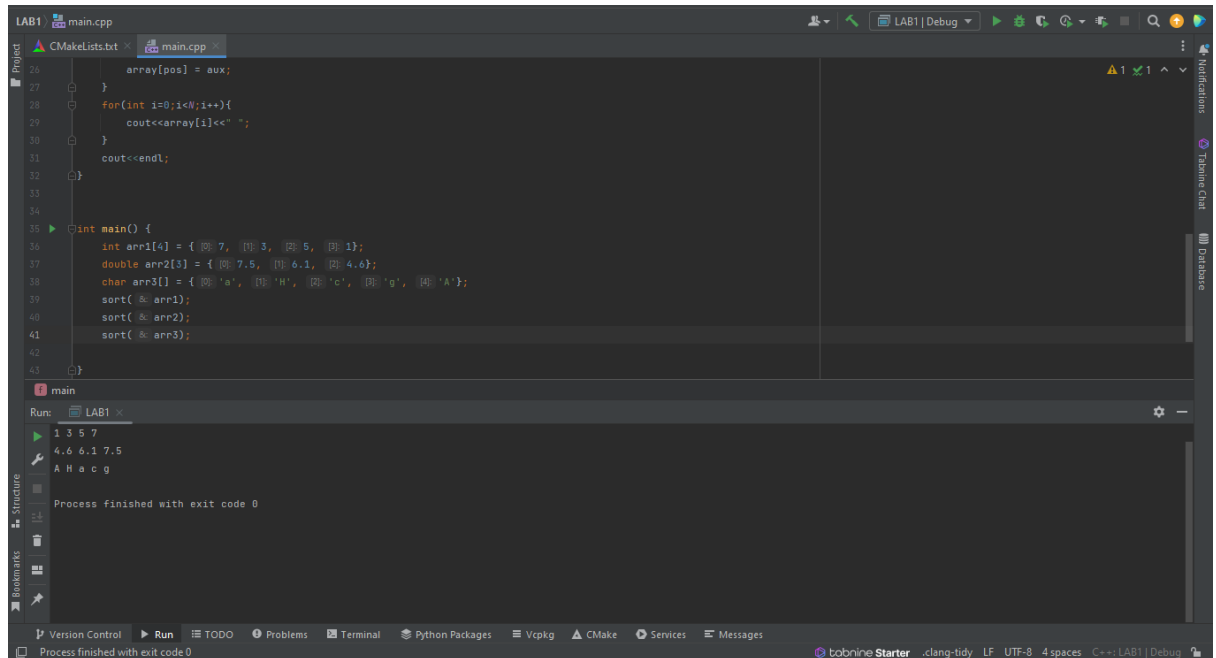
    return 0;
}
```

2.5 Descripción

La función sort toma como entrada un array y lo ordena en orden ascendente. El código utiliza un bucle for para iterar a través del array y encontrar la posición correcta para insertar cada elemento en el array ordenado. Si se encuentra una posición correcta, se inserta el elemento en esa posición. Se va verificando si el número anterior al actual es mayor, si es mayor se hace el cambio, se usa una variable auxiliar, para guardar el valor de la posición actual[5], si el valor de la posición anterior es mayor, entonces se le asigna ese valor en la posición [5], luego se le asigna el valor de aux en la posición[4] usando pos - -. Nuevamente usamos template para que haga varios tipos de datos, Finalmente, la función imprime el array ordenado.

En el main se crean tres arrays de diferentes tipos de datos y llama a la función sort para cada uno de ellos. Luego, imprime el array ordenado. El primer array es de tipo int, el segundo es de tipo double y el tercero es de tipo char.

2.6 Resultados



The screenshot shows a C++ IDE with a project named 'LAB1'. The main.cpp file contains the following code:

```
26     array[pos] = aux;
27 }
28 for(int i=0;i<N;i++){
29     cout<<array[i]<<" ";
30 }
31 cout<<endl;
32 }
33
34
35 int main() {
36     int arr1[4] = { [0] 7, [1] 3, [2] 5, [3] 1};
37     double arr2[3] = { [0] 7.5, [1] 6.1, [2] 4.6};
38     char arr3[] = { [0] 'a', [1] 'H', [2] 'c', [3] 'e', [4] 'A'};
39     sort(&arr1);
40     sort(&arr2);
41     sort(&arr3);
42 }
43
```

The Run window shows the output of the program:

```
1 3 5 7
4.6 6.1 7.5
A H a c g
```

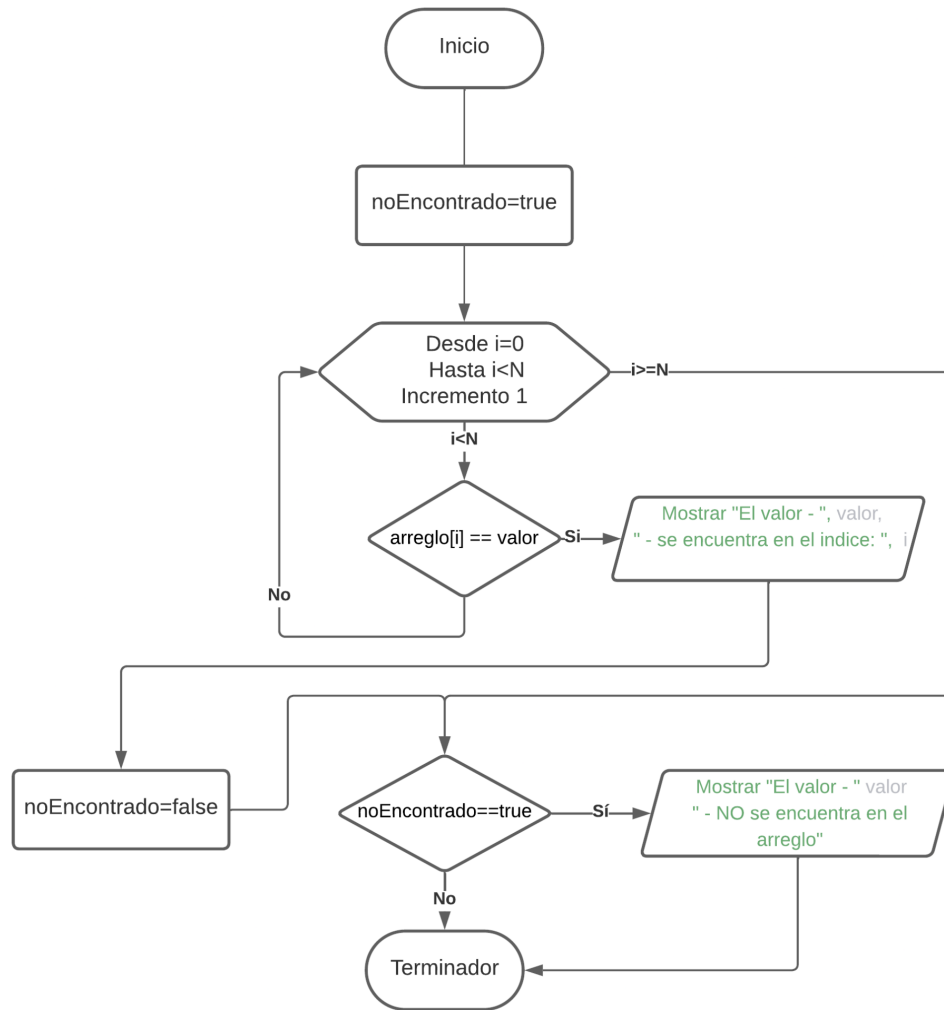
The process finished with exit code 0.

3. Punto 3

3.1 Enunciado

Escriba una función template para encontrar un valor en un array. Pruebe la función con dos arrays de tipo int y char.

3.2 Diagrama de flujo



3.3 Pseudocódigo

Algoritmo Ejercicio_3

Funcion findValue(arreglo,valor)

Definir noEncontrado Como Logico;

noEncontrado = Verdadero;

Para i=0 Hasta N-1 Con Paso 1 Hacer

si arreglo[i] = valor Entonces

Escribir "El valor - ", valor, " - se encuentra en el indice: ", i;

noEncontrado = Falso;

romper;

FinSi

FinPara

si noEncontrado=Verdadero Entonces

Escribir "El valor - ", valor, " - NO se encuentra en el arreglo";

FinSi

FinFuncion

Definir x Como Entero;

Definir y Como Caracter;

arrayInt[4] = {1,3,6,8};


```

arrayChar[4] = {'a','b','c','d'};
x = 3;
findValue(arrayInt,x);
x = 8;
findValue(arrayInt,x);
x = 10;
findValue(arrayInt,x);
y = 'b';
findValue(arrayInt,y);
y = 'c';
findValue(arrayInt,y);
y = 'T';
findValue(arrayInt,y);
FinAlgoritmo

```

3.4 Código fuente

```

template<typename T, int N>
void findValue(T (&arreglo)[N], T valor){
    bool noEncontrado = true;
    for(int i=0; i<N; i++){
        if(arreglo[i] == valor){
            cout << "El valor - " << valor
                << " - se encuentra en el indice: "
                << i << endl;
            noEncontrado=false;
            break;
        }
    }
    if(noEncontrado){
        cout << "El valor - " << valor
            << " - NO se encuentra en el arreglo" << endl;
    }
}

int main() {
    int arrayInt[] = {1,3,6,8};
    char arrayChar[] = {'a','b','c','d'};
    findValue(arrayInt,3);
    findValue(arrayInt,8);
    findValue(arrayInt,10);
    findValue(arrayChar,'b');
    findValue(arrayChar,'c');
    findValue(arrayChar,'T');
    return 0;
}

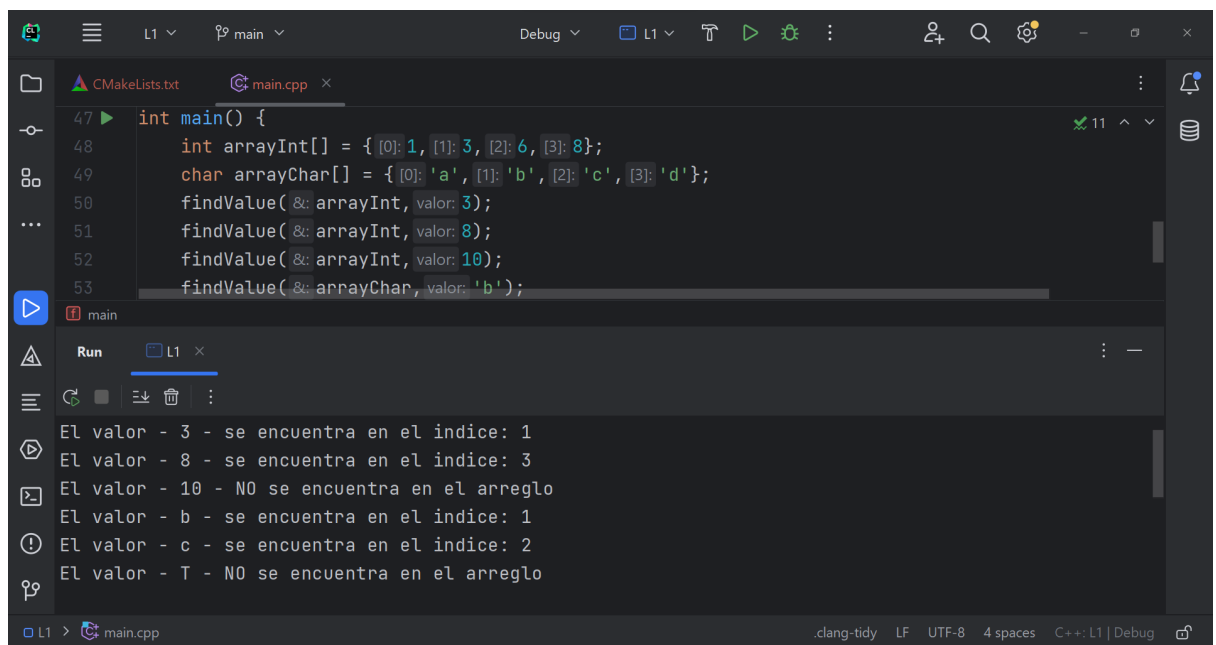
```

3.5 Descripción

La función findValue recibe como parámetros una referencia de un arreglo de tipo “T” y de tamaño “N”; el tipo “T” es una plantilla o bien llamada “template” que nos

permite recibir en este caso, arreglos que están compuestos de cualquier tipo de datos. Además, recibe un valor también de tipo "T", el cual será el elemento que se va a buscar en el arreglo. La función comienza inicializando una variable booleana en "true", luego se inicia un ciclo for, el cual nos permite iterar en todos los elementos del arreglo. Luego, con el condicional, podemos saber si el elemento que buscamos llamado "valor", se encuentra en dicho arreglo. Si se encuentra en el arreglo, se muestra por pantalla el índice en el cual está ubicado, nuestra variable booleana toma el valor del "falso" y se rompe el ciclo con el "break". De lo contrario, si no se encuentra en el arreglo, el programa sale del ciclo y entra en el siguiente condicional por medio de la variable booleana declarada al principio de la función, el condicional muestra un texto por pantalla que indica que el elemento no se encuentra en el arreglo.

3.6 Resultados



The screenshot shows a C++ IDE with a file named `main.cpp` open. The code defines two arrays: `arrayInt` with values {1, 3, 6, 8} and `arrayChar` with values {'a', 'b', 'c', 'd'}. It then calls a `findValue` function for each element. The output window shows the results of these searches.

```
47 int main() {
48     int arrayInt[] = { [0]: 1, [1]: 3, [2]: 6, [3]: 8 };
49     char arrayChar[] = { [0]: 'a', [1]: 'b', [2]: 'c', [3]: 'd' };
50     findValue(&arrayInt, valor: 3);
51     findValue(&arrayInt, valor: 8);
52     findValue(&arrayInt, valor: 10);
53     findValue(&arrayChar, valor: 'b');
```

Run

```
El valor - 3 - se encuentra en el indice: 1
El valor - 8 - se encuentra en el indice: 3
El valor - 10 - NO se encuentra en el arreglo
El valor - b - se encuentra en el indice: 1
El valor - c - se encuentra en el indice: 2
El valor - T - NO se encuentra en el arreglo
```

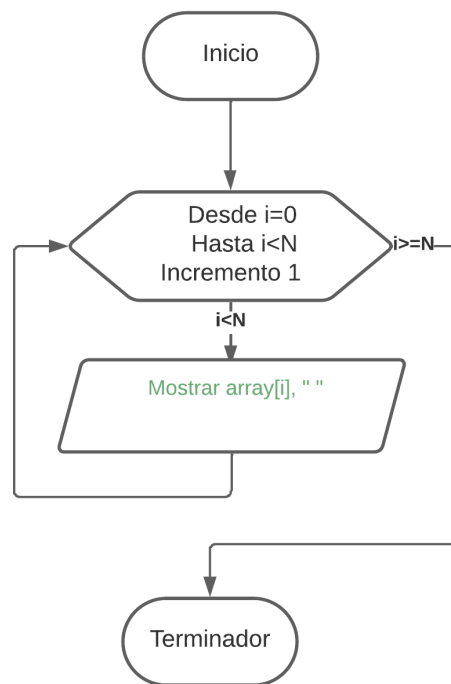
4. Punto 4

4.1 Enunciado

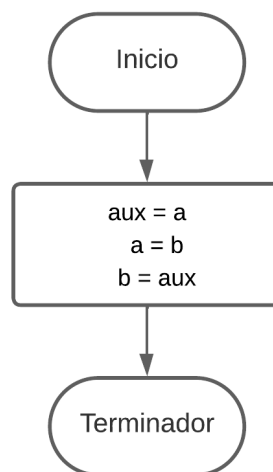
Defina una función para revertir el orden de los elementos de un array de cualquier tipo. Pruebe el programa con un array de `int`, `double`, `char` y `string`. Use una función auxiliar para intercambiar dos elementos cualesquiera. Use una función auxiliar para imprimir el contenido de un arreglo antes y después del intercambio.

4.2 Diagrama de flujo

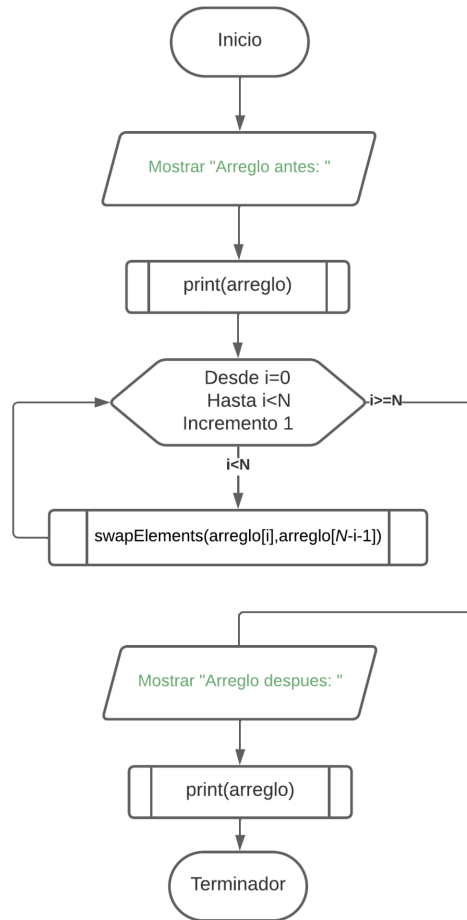
Función print:



Función swapElements:



Función reverseArray:



4.3 Pseudocódigo

Funcion print(arreglo)

Para i<-0 Hasta N-1 Con Paso 1 Hacer

 Escribir arreglo[i];

FinPara

 Escribir "";

FinFuncion

Funcion swapElements(a,b)

 aux = a;

 a = b;

 b = aux;

FinFuncion

Funcion reverseArray(arreglo)

 Escribir "Arreglo antes: ";

 print(arreglo);

 Para i<-0 Hasta N/2 Con Paso 1 Hacer

 swapElements(arreglo[i], arreglo[N-i-1]);

 FinPara

 Escribir "Arreglo despues: ";

 print(arreglo);

FinFuncion

Algoritmo Ejercicio_4

```
arrayInt[4] = {1,3,6,8};
arrayChar[4] = {'a','b','c','d'};
arrayDouble[5] = {234.5,23534.2,1.45,89.2266,1.1};
arrayString[5] = {"Santiago","Juan","Camilo","Dhanielt","Sofia"};
reverseArray(arrayInt);
reverseArray(arrayDouble);
reverseArray(arrayChar);
reverseArray(arrayString);
```

FinAlgoritmo

4.4 Código fuente

```
template<typename T, int N>
void print(T(&array) [N]) {
    for(int i=0; i < N; i++){
        cout << array[i] << " ";
    }
    cout << endl;
}

template<typename T>
void swapElements(T &a, T &b) {
    T aux = a;
    a = b;
    b = aux;
}

template<typename T, int N>
void reverseArray(T (&arreglo) [N]) {
    cout << "Arreglo antes: ";
    print(arreglo);
    for(int i=0; i<N/2; i++){
        swapElements(arreglo[i],arreglo[N-i-1]);
    }
    cout << "Arreglo despues: ";
    print(arreglo);
}

int main() {
    int arrayInt[] = {1,3,6,8};
    char arrayChar[] = {'a','b','c','d'};
    double arrayDouble[] = {234.5,23534.2,1.45,89.2266,1.1};
    string arrayString[] = {"Santiago","Juan", "Camilo",
"Dhanielt", "Sofia"};

    reverseArray(arrayInt);
    reverseArray(arrayDouble);
```

```

reverseArray(arrayChar);
reverseArray(arrayString);
return 0;
}

```

4.5 Descripción

El código consta de tres funciones, la primera llamada “print”, nos permite mostrar por pantalla el contenido del arreglo, dicha función recibe la referencia de un arreglo de tipo “T” y de tamaño “N”, de la misma forma que anteriormente, “T” es un “template” que nos permite recibir un arreglo de cualquier tipo de dato; la función utiliza un ciclo for para iterar en cada uno de los elementos del arreglo y posteriormente los imprime. La segunda función llamada “swapElements”, nos permite intercambiar dos elementos cualquiera, esto por medio del “template” llamado “T”; la función recibe como parámetros dos elementos y por medio de una variable auxiliar, se intercambia su contenido. Por último, la tercera función llamada “reverseArray”, nos permite cambiar el orden de los elementos de un arreglo; dicha función recibe como parámetros la referencia de un arreglo de cualquier tipo de datos, esto gracias nuevamente al “template T” y además, recibe el tamaño del arreglo. La función primero muestra por pantalla como está el arreglo antes de invertirlo, por medio de la función “print”; luego hace uso del ciclo for para iterar en la primera mitad de elementos del arreglo, para después hacer uso de la función “swapElements” y de esta manera ir intercambiando los elementos de la primera posición con la última, de la segunda posición con la penúltima y así sucesivamente hasta que los invierta por completo, después, la función imprime los elementos del arreglo que ahora está invertido, por medio de la función “print”. Si el arreglo tiene un número de elementos impar, habrá un elemento para el cual su posición no cambiará, dicho elemento es el del centro.

4.6 Resultados

```

main.cpp
47 int main() {
48     int arrayInt[] = { [0]: 1, [1]: 3, [2]: 6, [3]: 8 };
49     char arrayChar[] = { [0]: 'a', [1]: 'b', [2]: 'c', [3]: 'd' };
50     double arrayDouble[] = { [0]: 234.5, [1]: 23534.2, [2]: 1.45, [3]: 89.2266, [4]: 1.1 };
51     string arrayString[] = { [0]: "Santiago", [1]: "Juan", [2]: "Camilo", [3]: "Dhaniel", [4]: "Sofia" };
52
53     // ... (function calls) ...
54 }

```

Run

```

Arreglo antes: 1 3 6 8
Arreglo despues: 8 6 3 1
Arreglo antes: 234.5 23534.2 1.45 89.2266 1.1
Arreglo despues: 1.1 89.2266 1.45 23534.2 234.5
Arreglo antes: a b c d
Arreglo despues: d c b a
Arreglo antes: Santiago Juan Camilo Dhaniel Sofia
Arreglo despues: Sofia Dhaniel Camilo Juan Santiago

```