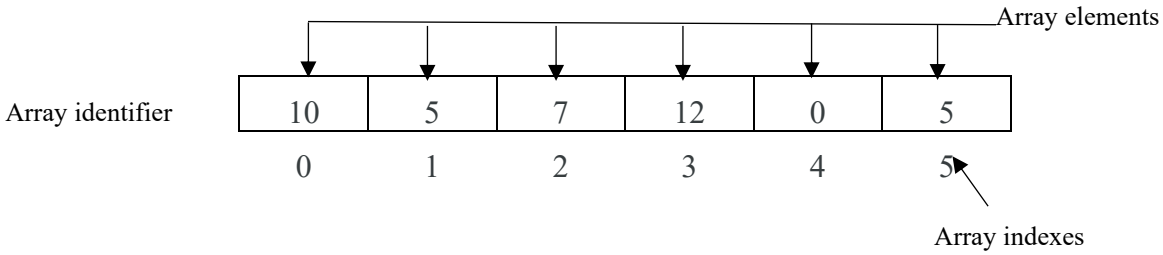
Chapter 5 : Array Structure

1. Introduction

An array is a data structure that consists of a set of values of the same data type with a single identifier name. Elements *are stored at contiguous memory locations.*

Basic terminologies of array

- **Array Index:** In an array, elements are identified by their indexes. Array index starts from 0.
- **Array element:** Elements are items stored in an array and can be accessed by their index.
- **Array Length:** The length of an array is determined by the number of elements it can contain.



2. Creating Arrays

Syntax for declaring an array:

Many options are possible to declare an array:

1st option

```
var   array_name : array[0.. number_of_values_to_hold-1] of data_type;
```

2nd option

```
var  array_name : array[number_of_values_to_hold] of data_type;
```

3nd option

```
Type name_of_type  : array[0..numberOfValuesToHold-1] of data_type;
Var   array_name : name_of_type ;
```

4th option

```
array_name [number_of_values_to_hold] : data_type
```

Example :

```
const numDays = 7 ; N = 5 ;
type numArr : array[0..4] of int ;

daysOfWeek : array[0..numDays-1] of string ;
vowels : array[0..N-1] of char;
scores : numArr ;
```

Once declared, arrays have to be initilazed / filled

Example :

```
daysOfWeek = [“sunday”, “monday”,”tuesday”,”wednesday”,”thuesday”,”friday,”saturday”]
vowels = [‘a’, ‘e’, ‘i’, ‘o’, ‘u’] ;
scores = [12, 8, 14, 9, 10] ;
```

3. Accessing Array Elements

An element from an array is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. The number inside the square brackets [] is called a **subscript**. It’s used to reference a certain element of the array.

3.1.Access to an element

Syntax To refer to a specific element in the array:

```
array_name[subscript number] ;
```

Example:

```
myDay = daysOfWeek[1] ;
```

This instruction assigns the value “monday” to the variable myDay.

3.2. Browsing arrays

Navigating through an array can be done in several ways, the most used are: The browsing from the first to the last element, and the browsing from the last to the first element.

3.2.1. Displaying elements of an array T with N elements

In order to go through all the elements of the array, we use a loop with an increment/decrement of the index from the first to the last element of the array or vice versa.

```
for i = 0 to N-1
  write(T[i])
endFor
```

or (the reverse order)

```
for i= (N-1) to 0
  print(T[i])
endFor
```

3.2.2. Reading elements of the array T with N elements

Example1 : Initializing the elements to 0

```
i = 0
while i < N-1
  begin
    T[i] ← 0;
     i++;
  end
endWhile
```

Example2 : Reading the elements

```
i = 0
while i < N-1
  begin
     write('Enter the ", i, " the element of the array : " );
     readln(T[i]);
     i++;
  end
endWhile
```

4. Array algorithms

4.1.Find the maximum and minimum element in an array (and their indexes)

```
algorithm max_min_tab;
const N=10;
var tab : array[0..N-1] of integer ;
    i, max, min, indexMin, indexMax : integer;

begin
   min ← tab[0] ;
   max ← tab[0] ;
  indiceMin ← 0 ;
   indiceMax ← 0 ;
   for  i ←  0 to  (N-1) do
       begin
          if (min > tab[i])
```

```
        then begin
                min ← tab[i] ;
                indiceMin ← i;
              end;
        endIf
        if (max < tab[i])
        then begin
                max← tab[i] ;
                indiceMax ← i;
              end;
        endIf
    endFor
    writeln("le min =", min,  " at the index : ",  indexMin) ;
    writeln("le max =", max, " at the index : ", indexMax) ;
end.
```

4.2.Find the sum and the product of all elements in an array

```
algorithm sum_product_tab;
const N=10 ;
var tab : array [0..N-1] of integer;
    i, sum, product : integer;
begin
  sum ← 0 ;
  product ← 1 ;
  for i ← 0 to (N-1) do
     begin
        sum ← sum + tab[i] ;
        product ← product * tab[i];
    end ;
  endFor
  writeln("The sum is" : , sum) ;
  writeln("The product is" : , product) ;
end.
```

4.3.Searching for elements in an array

There are two main algorithms for searching for elements in an array: linear search and binary search.

- Linear search is a simple algorithm that iterates over the array, comparing each element to the target element. If the target element is found, a Boolean variable is set to true, otherwise it is false. Given the array tab having as length N, and the element to search x:

```
found ← false ;
i ← 0 ;
while ((found = false) and (i<N)) do
    begin
        if  tab[i] = x   then begin found ← true ;
        endIf ;
        i ← i+1 ;
     end;
endWhile
```

- Binary search is a more efficient algorithm that can be used on sorted arrays. It works by repeatedly dividing the array in half and comparing the middle element to the target element. If the target element is equal to the middle element, the algorithm returns its index. Otherwise, the algorithm recursively searches the half of the array that contains the target element.

4.4.Shift array

4.4.1. Left shift with insertion of 0

```
algorithm left_shift_zeroInsert_tab;
const N=10;
var tab : array[0.. N-1]  of integer ;
     i : integer;
begin
  // read the array
   ....
   for i ← 0 to (N-1-1) do
          tab[i]  ← tab[i+1];
    endFor;
    tab[N-1] ← 0;
end.
```

4.4.2. Right Circle shift without insertion

```
algorithm right_shift_zeroInsert_tab;
const N=10;
var   tab : array[0.. N-1] of integer ;
      i , tmp : integer;
begin
  // read the array
  ...
```

```
    tmp ← tab[N-1]
    for i ← (N-1) to 1 do
        tab[i]  ← tab[i-1];
    endFor;
    tab[0] ← tmp;
end.
```

5. Sorting Arrays

Sorting is one of the basic operations on the arrays. Indeed; it's usually helpful when we have an array to be able to put it in some sort of order (be it numerical or alphabetical).

There are many algorithms to sort a list. Some are simple and some are complex. Some are fast while others are slow. We'll look at a few of the most common ones.

5.1.Selection sort

Selection sort is a simple sorting algorithm that works by repeatedly finding the smallest element in an unsorted array and swapping it with the first element in the array. This process is repeated until the entire array is sorted.

Illustration : Imagine you have a list of number (characters, strings, etc…)

8 4 2 5 3 7

Look through the list and find the smallest. Exchange the smallest with the first item in the list.

2 | 4 8 5 3 7

Now look through everything to the right of the | for the smallest. Exchange the smallest with the first item in the list to the right of the |

2 3 | 8 5 4 7

Repeating the steps results in:

2 3 4 | 5 8 7

2 3 4 5 | 8 7

2 3 4 5 7 | 8

2 3 4 5 7 8 |

```
for i ← 0 to (N – 1-1) do
 Begin
   smallest = i ;
   for j ← (i + 1) to N-1 do
      if array[j] < array[smallest] then    smallest = j
   endFor ;
   temp = array[i]
   array[i] = array[j]
   array[j] = temp
```

```
 end
endFor
```

5.2.Bubble Sort

Bubble sort is a simple sorting algorithm that works by repeatedly comparing adjacent elements in an array and swapping them if they are in the wrong order. Thus, the smallest element "bubbles" to the top of the array. This process is repeated until the entire array is sorted.

```
for i ← 0 to n-1-1 do
   begin
     for j ← 0 to n-i-1 do
        if  (tab[j]>tab[j+1])
        then  begin
                    tmp ←tab[j];
                    tab[j] ← tab[j+1];
                    tab[j+1] ←tmp;
                end;
        endIf;
    endFor ;
end ;
endFor.
```

Illustration

Array to sort : 8 4 2 5 3 7

Pass 1

Compare 8 and 4: Swap

Array: 4 8 2 5 3 7

Compare 8 and 2: Swap

Array: 4 2 8 5 3 7

Compare 8 and 5: Swap

Array: 4 2 5 8 3 7

Compare 8 and 3: Swap

Array: 4 2 5 3 8 7

Compare 8 and 7: swap

Array: 4 2 5 3 7 8

Pass 2

Compare 4 and 2: Swap

Array: 2 4 5 3 7 8

Compare 4 and 5: No swap required

Array: 2 4 5 3 7 8

Compare 4 and 3: Swap

Array: 2 3 4 5 7 8

Compare 4 and 7: No swap required

Array: 2 3 4 5 7 8

Compare 4 and 8: No swap required

Array: 2 3 4 5 7 8

Pass 3

Compare 2 and 3: No swap required

Array: 2 3 4 5 7 8

Compare 2 and 4: No swap required

Array: 2 3 4 5 7 8

Compare 2 and 5: No swap required

Array: 2 3 4 5 7 8

Compare 2 and 7: No swap required

Array: 2 3 4 5 7 8

Compare 2 and 8: No swap required

Array: 2 3 4 5 7 8

5.3.Insertion Sort

Insertion sort is a simple sorting algorithm that works by inserting each element in an unsorted array into its correct position in a sorted array. In other words, as an element is added to the list, it is **inserted** into the correct position.

```
for i ← 1 to N-1 do
   temp = array[i] ;
   j = i – 1 ;
   while j >= 0 and array[j] > temp do
     begin
        array[j + 1] = array[j] ;
        j = j – 1 ;
      end ;
   array[j + 1] = temp ;
```

6. 2-Dimensional arrays (matrices)

The elements of an array may be of any type. They could even be an array. This type of array is known as a **multi-dimensional array**.

One-dimension arrays are called vectors. When an array has 2 or more dimensions, it is called a matrix.

	1st column	2nd column	…	..	Mth column
1st row					
2nd row					
…					
Nth row					

6.1. Elements of the matrices

Syntax of declaration:

array_name : array[0 ..numberOfRows-1][0 ..numberOfColumns-1] of data_type;

Accessing elements of a matrix

To refer to a specific element in the array, must specify two subscripts.

Syntax:

array_name [i][j]

Example : Initilization of elements to 0

```
algorithm init_mat ;
const  n=4 , m=5;
var      mat : array [0 ..n-1][0 .. m-1] of integer ;
          i, j : integer;

begin
  for  i ← 0 to (n -1) do
     for j ← 0 to (m-1) do mat[i][j] ← 0;
     endFor;
endFor;
end.
```

6.2. Operations on matrixes

6.2.1. Sum of elements of a matrix

```
algorithm sum_mat ;
const n=4, m=5;
var     mat : array[0 ..n-1][0 ..m-1] of integer ;
        i, j, sum : integer ;
```

```
begin
   for i ← 0 to (n -1) do
      begin
         sum ← 0 ;
         for j ← 0 to (m-1) do sum ← sum + mat[i][j] ;
         endFor;
         writeln(sum);
       end;
   endFor;
end.
```

6.2.2. Sum of two matrices

To sum two matrices, matrices should have the same dimensions.

```
algorithm sum_2_mat ;
const n=4, m=5 ;
type  matrice : array [0 ..n-1][0 ..m] of integer ;
var   mat1, mat2, mat3 : matrice ;
      i,  j : integer;

begin
   for i ← 0 to (n-1) do
       for j ← 0 to (m-1) do
           mat3[i][j] ← mat1[i][j] + mat2[i][j] ;
       endFor;
   endFor;
end.
```

6.2.3. Product of two matrices

To perform the product of two matrices, if the first matrix is (N,M) dimension, the second should be (M, P).

```
algorithm product_mat;
const n=4,m=5,p=6;
var mat1 : array[0.. n][0 ..m] of integer ;
    mat2 : array[0 ..m][0 ..p] of integer ;
    mat3 : array[0.. n][0 ..p] of integer ;
    i, j, k, sum : integer ;  // the variable sum is used to compute the product between vectors

begin
  for k ← 0 to (n-1) do
      for i ← 0 to (p-1) do
          begin
             sum ← 0;
             for j ← 0 to (m-1) do sum ← sum+(mat1[k][j]*mat2[j][i]);
```

```
            endFor ;
            mat3[k][i] ← sum;
          end ;
      endFor;
   endFor ;
end.
```

7. Arrays in C programs

In C programs, subscripts start from 0 and run to N-1 (where N is the value within the square brackets).

7.1. Creating an array in C

data_type array_name[array_size];

Example : int tab[10] ;

7.2. Operations on arrays

A. Searching for an element in an array

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

int main() {
  int arr[] = {1, 3, 5, 7, 9};
  int target = 5;
  int i;
  bool found = false;


  for (i = 0; i < sizeof(arr)/ sizeof(arr[0]); i++) {

    if (arr[i] == target) {
      found = true;
      break;
    }
  }

  if (found) {
    printf("The element %d is found at index %d\n", target, i);
  } else {
    printf("The element %d is not found in the array\n", target);
  }
    return 0;
}
```

B. Sorting

The following program performs the sorting according to the selection sort algorithm. This program works by iterating over the array and finding the smallest element in the unsorted part of the array. It then swaps the smallest element with the current element. The program repeats this process until the entire array is sorted.

```c
#include <stdio.h>
int main() {
  int arr[] = {5, 2, 8, 7, 1};
  int n = 5;
  int i, j, temp, smallest ;

  for (i = 0; i < n - 1; i++) {
    smallest=i;
    for (j = i + 1; j < n; j++) {
      if (arr[j] < arr[smallest]) {
       smallest = j;
      }
    }
    temp = arr[i];
    arr[i] = arr[smallest];
    arr[smallest] = temp;
  }

  printf("The sorted array is:\n");
  for (i = 0; i < n; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");

  return 0;
}
```

Operations on Matrices:

The sum of the elements of the diagonal

```c
#include <stdio.h>

int main() {
  int matrix[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
  int i, sum = 0;

  for (i = 0; i < 3; i++) {
    sum += matrix[i][i];
  }

  printf("The sum of the diagonal elements is %d\n", sum);

  return 0;
}
```