M'hamed Bougara University - Boumerdes
Faculty of Sciences
Computer Science Department

Algorithms and Data Structures 1

L1 Computer Science

Dr RAHMOUNE Nabila

2023/2024

Chapter 3
Arrays and Character Strings

Outline

- Introduction
- Unidimensional array
- Bidimensional array
- Characters string
- Applications

Arrays structure

Definitions

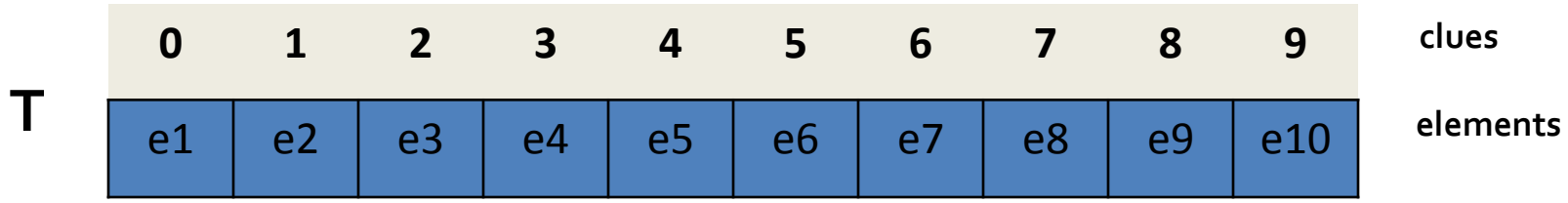
- An array is a grouping of variables of the same type , it is identified by a name . Each of the variables in the array is numbered, this number is called an index .

- Each variable of the table is therefore characterized by the name of the table and its index.

Arrays structure

Definitions

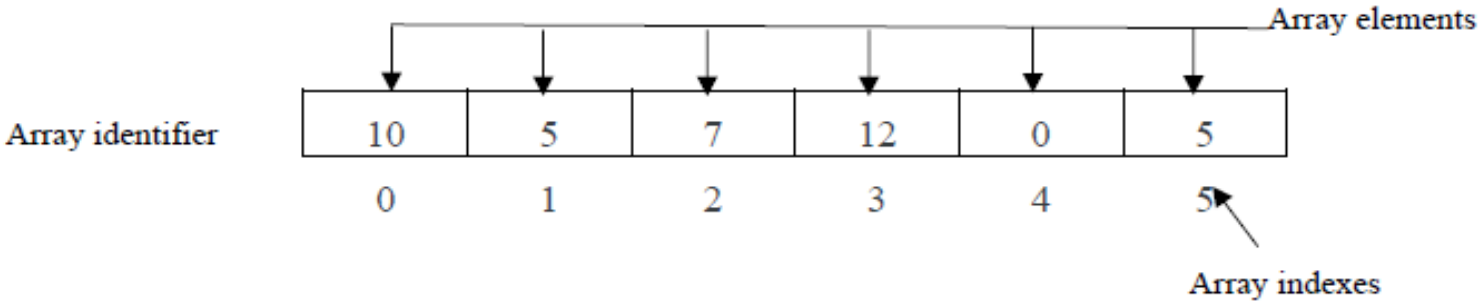
- If for example, T is an array of 10 variables , then each of them will be numbered and it will be possible to find it using **the name of the array and the index of the variable simultaneously.**
- The different variables of T will have numbers from 0 to 9, and we will call each of these variables **an element of T** .



Arrays structure

Static Representation

The array is stored in memory as a sequence of contiguous cells. Once created, new cells cannot be added to or removed from the array (static).



Arrays structure

Declaration

- As the variables of an array must be of the same type, this type should be specified when declaring the array. Similarly, when declaring the array, specify the number of variables it contains.

- The syntax is:

  <arrayName> : array[<size>] of <type>**;**

For example,

T: array [4] of integer; **//** declares an array T containing 4 integer type variables.

Arrays structure

Assignment

Assigning a value x to an element i of a numeric array T is done by:

T[i] ← x;

For example, the statement:

T[0] ← 5; assigns the value 5 to the first element of the array T.

Arrays structure

Access to Elements

- The elements of an n-element array are indices from 0 to n-1. We denote by T[i] the element of index i of the array T.
- The five elements of the table in the example above are therefore noted

  T[0], T[1], T[2], T[3] and T[4].

Arrays structure

Reading an array

To fill an array of N numbers, we use "**read**" N times like:

read (tab[0])
read (tab[1])
…
read (tab[N-1])

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

or the reverse order

```
for (i← N-1) to 0
read(tab[i]);
endFor
```

Arrays structure

Displaying an array

We substitute the read operation with the write operation.

```
write (tab[0])
write (tab[1])
…
write (tab[N-1])
```

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

or the reverse order

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

Arrays structure

Example

Write an algorithm that receives the averages of N students, where N is determined by the user, then calculates the number of students who failed the subject (average less than 10).

```
Algorithm nb_adjourned
var avg :array(100) of real;
 i, aj, N : integer
Begin
print ("enter number of students (<", MAX, ")")
read (N)
for (i ← 0 to N-1) do
read (avg[i]);
endfor
aj ← 0
for i ← 0 to N-1 do
 if  (avg[i]<10)  then
aj ← aj+1;
end if
 end for
print("the number of adjourned is ", aj)
End.
```

Arrays structure in language C

Declaration

- The syntax in C is:

<type> **<tablename>** [<size>] **;**

For example: **int T[4];**

- We will declare an array E like this:

int E[10];

The above statement is for an array of 10 ints called E

- It is then necessary to enter the 10 values.

Arrays structure in language C

Initialization

- It is possible to initialize the elements of an array at the declaration, we do this as for scalar variables:

<type> <name>[<size>] = <initialization value>;

- The only thing that changes is the way of writing the initialization value, we write in braces all the elements of the array, we arrange them in order of increasing index, separating them with commas.

Arrays structure in language C

Initialization

The general syntax of the initialization value is therefore:
<type> <name> [<size>] = { <value 0 >, <value 1 >, . . . , <value n -1> };

For example, we create an array containing the first 5 odd numbers like this:

int T[5] = {1, 3, 5, 7, 9};

Arrays structure in language C

Access to Elements

- The elements of an n-element array are indices from 0 to n-1. We denote by **T[i]** the element of index i of the array T.
- The five elements of the table in the example above are therefore noted

T[0], T[1], T[2], T[3] and T[4].

Arrays structure in language C

```
printf ( "Enter ten values: \n" );
printf ( "1" );
sc anf (%d" , &E [ 0 ] );
printf ( "2" );
sc anf (%d" , &E [ 1 ] );
printf ( "3" );
sc anf (%d" , &E [ 2 ] );
printf ( "4" );
sc anf (%d" , &E [ 3 ] );
printf ( "5" );
sc anf (%d" , &E [ 4 ] );
printf ( "6" );
sc anf (%d" , &E [ 5 ] );
printf ( "7" );
sc anf (%d" , &E [ 6 ] );
printf ( "8" );
sc anf (%d" , &E [ 7 ] );
printf ( "9" );
sc anf (%d" , &E [ 8 ] );
printf("10");
sc anf (%d" , &E [ 9 ] );
```

```
printf ( "Enter ten values: \n" );
for (i = 0; i < 10; i++)
{
printf ( "%d" , i +1);
sc anf ( "%d" , &E[ i ] ) ;
}
```

This type of loop is called **an array traversal** . As a general rule, **loops are used to handle arrays** , these allow you to perform a process on each element of an array.

Bidimensional array

Definitions

The matrix is represented in memory by a sequence of adjacent cells. A cell cannot be removed or added to the matrix after its creation (static).
Example of a matrix with 4 rows and 5 columns of real numbers.

	0	1	2	3	4
0	15	7	-3	0	9
1	6	12	4	33	85
2	2	-8	17	28	-52
3	14	42	36	49	-12

Bidimensional array

Declaration

<matrixName>: Array[Rows][Columns] **of** <elementType>;
<matrixName>: The identifier given to the matrix (variable name).
Rows: Number of rows.
Columns: Number of columns.

Example :

```
const R=100 ;
const C=100;
M : Array [R][C] of real ;
mat1,mat2 : array[30][20] of integer
```

Bidimensional array

Access to Elements

- To access a single element of the matrix, we use the matrix name with an index inside two brackets `[` and `]` or ( ) specifying the row number, and another index inside two brackets `[` and `]` specifying the column number.
- To access the element in the first row and first column of matrix mat, we use mat[0][0] or mat[1][1].

Syntax:

array_name [i][j]

Bidimensional array

Reading a matrix

To fill the matrix `M[Rows][Columns]`, we fill in `Rows` rows. Since each row is a one-dimensional array with `Columns` elements.

For (j ← 0 to C-1)
do
read(M[0][j])
end for
	For (j← 0 to C-1)
	do
	read(M[1][j])
	endfor
		for (j← 0 to C-1)
		do
		read(M[L-1][j]) ;
		endFor

Bidimensional array

Reading a matrix

To fill the matrix `M[Rows][Columns]`, we fill in `Rows` rows. Since each row is a one-dimensional array with `Columns` elements.

For (j ← 0 to C-1)
do
read(M[0][j])
end for
	For (j← 0 to C-1)
	do
	read(M[1][j])
	endfor
		for (j← 0 to C-1)
		do
		read(M[L-1][j]) ;
		endFor

Bidimensional array

Reading a matrix

We can fill all the matrix using double loop

```
For ( i← 0 to L -1) do
  For ( j← 0 to C-1) do
   read(M[i][j]);
  endfor;
enfor;
```

Bidimensional array

Displaying an array

Similar to reading, the `write` statement is repeated.

```
for (i← 0 to L-1) do
    for (j← 0 to C-1 ) do
  print (M[i][j]); // write(M[i][j])
   endFor
 endFor
```

Bidimensional arrays in language C

Declaration

- The syntax in C is:

<type> **<tablename>** [<size1>] [<size2>] **;**

For example: **int T[4] [5];**

Arrays structure in language C

Initialization

- It is possible to initialize the elements of an array at the declaration, we do this as for scalar variables:

<type> <name>[<size1>] [<size2>]= <initialization value>;

- The only thing that changes is the way of writing the initialization value, we write in braces all the elements of the matrix, we arrange them in order of increasing index, separating them with commas for the same arrow and with scoler ”;” for the clommun.

Arrays structure in language C

Initialization

The general syntax of the initialization value is therefore:
<type> <name> [<size>] = { <value 0 >, <value 1 >, . . . , <value n -1> ; <value 0 >, <value 1 >, . . . , <value n -1>; …… ; <value 0 >, <value 1 >, . . . , <value n -1> };

For example, we create an array with 2 arrows and 5 colomuns :

int T[5][2] = {1, 3, 5, 7, 9; 8,6,0,2,1};

Arrays structure in language C

Access to Elements

- The elements of an n-element array are indices from 0 to n-1 for eatch arrow. We denote by **T[i] T[j]** the element of index i (arrow i) and index j (clomun j) of the array T.

- The five elements of the first arrow :

  T[0][0], T[0][1], T[0][2], T[0][3] and T[0][4].

Bidimensional array

Example: Sum of elements of a matrix

```
algorithm sum_mat ;
const n=4, m=5;
Var   mat : array[n-1][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;
       print(sum);
     endfor;
end
```

Character String

Definition

A character string, or simply a string, is a sequence of characters, like letters, numbers, or symbols that are grouped together. It's commonly used in programming to represent text data.

For example, "hello" and "123abc" are both strings.

- A string is a char array containing a null character.
- The null character has 0 for ASCII code and is written **'\0'**.
- The significant values of the character string are all those placed before the null character.

Character String

Declaration

<NameStringe> [<sizeMaxString>] : string ;

In C :

char < NameStringe > [<sizeMaxString >] ;

Example

C: array[200] of char;

Character String

Initializing a string

- The initialization of a string can be achieved either through the conventional array syntax.
- The last character indicates the end of the string, or by using quotation marks.

Character String

Example

```
#include <stdio.h>
int main() {
   // Initializing a string using array syntax
   char string1[] = {'H', 'e', 'l', 'l', 'o', '\0'}; // '\0' indicates the
            end of the string

 // Initializing a string using quotation marks
   char string2[] = "World";
   printf("String 1: %s\n", string1);
   printf("String 2: %s\n", string2);
   return 0;
}
```

Character String

Reading and writing a string

- Reading of the string stops as soon as it encounters a separator character.
- Newline, space, tabulation, etc.

- **In C** writing a string is done as the example following:

Character String

Reading and writing a string

```
#include <stdio.h>

int main() {
    char str[100]; // Declare a character array to store the string

    // Reading a string from the user
    printf("Enter a string: ");
    scanf("%s", str); // Read a string from the user and store it in 'str'
        // Writing the string
    printf("You entered: %s\n", str); // Print the string

    return 0;
}
```

Character String

Operations on strings in C

- Strings can be manipulated like arrays.
- We can apply our knowledge of manipulating simple arrays as discussed earlier.
- Another approach is to use the ***string.h*** library provided by the compiler, which offers a set of predefined functions to assist in string manipulation.

Character String

String Concatenation: Combining two strings into one.

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

int main() {
    char str1[50] = "Hello";
    char str2[50] = " World";
    strcat(str1, str2); // Concatenate str2 to str1
    printf("Concatenated String: %s\n", str1);
    return 0;
}
```

Character String

String Comparison: Comparing two strings.

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

int main() {
    char str1[] = "apple";
    char str2[] = "banana";
    int result = strcmp(str1, str2); // Compare str1 and str2
    if (result < 0)
        printf("str1 is less than str2\n");
    else if (result > 0)
        printf("str1 is greater than str2\n");
    else
        printf("str1 is equal to str2\n");
    return 0;  }
```

Character String

String Copying: Copying the contents of one string to another.

```
#include <stdio.h>
#include <string.h>
int main() {
    char src[] = "Copy me!";
    char dest[50];
    strcpy(dest, src);     // Copy src to dest
    printf("Copied String: %s\n", dest);
    return 0;
}
```

Character String

String Length: Finding the length of a string.

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

int main() {
    char str[] = "Hello, World!";
    int length = strlen(str); // Get the length of str
    printf("Length of String: %d\n", length);
    return 0;
}
```