Ministry of Higher Education and Scientific Research
Djilali BOUNAAMA University - Khemis Miliana(UDBKM)
Faculty of Matter Science and Computer Science
Department of Mathematics

Chapter : 6

Arrays and Strings

MI-L1-UEF121 : Algorithms and Data Structures I

Ali Khalfi

Khalfiali.udbkm@gmail.com

Course Topics

1. Problem and Introduction
2. One Dimensional Arrays: Vectors
   - 2.1 Definitions
   - 2.2 Declaration and manipulation
   - 2.3 Array Algorithms
3. Two-dimensional arrays: Matrixes
   - 3.1 Definitions
   - 3.2 Declaration and manipulation
4. Strings

Problem

Problem

Calculate the average of students, then display the final ranking?

- ✓ ***Suggestion 1*** : Use a repetitive structure that will read, calculate and display each student's average.
  - ✓ *Problem 1* : we cannot classify the students because we need all the averages already calculated.
- ✓ ***Suggestion 2*** : Use as many variables as students.
  - ✓ *Problem 1* : difficult to do and manage: we will write as many blocks of instructions as number of students.
  - ✓ *Problem 2* : we may not know in advance the exact number of students.

Introduction

- ✓ To remedy these problems, we must use a **data structure** capable of bringing all these variables together into one single variable.
- ✓ Such a structure is called an ***Array*** or a ***Table***. It's a **Structured Type**.
- ✓ The types presented so far are simple types. There are other so-called structured (complex) types.
- ✓ A structured type is any type defined on the basis of other types

Definition

- ✓ An array is a **data structure** allowing a set of elements of the same type to be **grouped** under the same name.
- ✓ An array is characterized by :
  - ➢ its **name**: a unique identifier
  - ➢ its **elements**: designates the ***boxes*** of the table. These elements have the same type which can be ***simple*** or ***structured***.
  - ➢ its **size**: the number of elements that the array can contain. Defined during declaration by a ***lower bound*** and ***upper bound***.

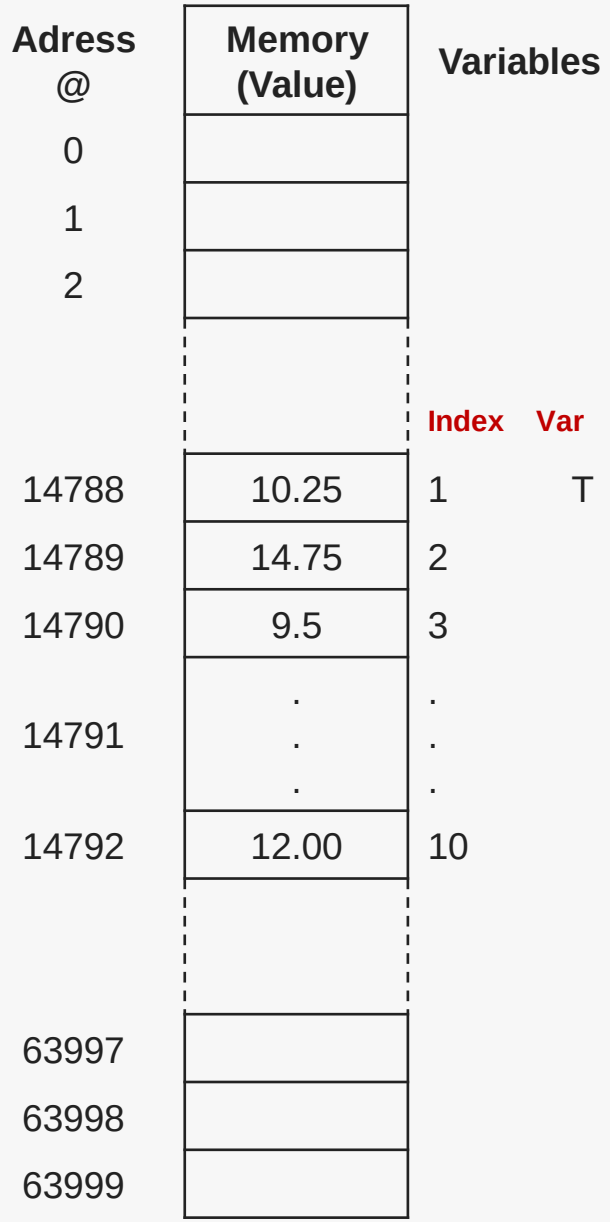
(the declared *maximum size* should not be confused with the *effective size* actually used in the algorithm)

Representation

- ✓ An array is made up of a number – fixed at the declaration – of ***contiguous*** boxes located in memory.
- ✓ An array is identified by the address of its ***1st*** element
- ✓ ***Access*** to an array element is done using its **index**.
- ✓ **Index** of an element: its ***index***, its ***position*** in the array

T

10.25	14.75	9.5	...	...	...	...	...	...	12.00
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**



1D Arrays:

Vectors

Definition

- ✓ A **one-dimensional array** or **vector** is a way of storing elements or values of the same type.
- ✓ It **groups** these elements in a fixed structure and allows access to each element via its **rank** or **index**.

A

4	12	8	-4	0	-7	15	2	-24	17
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Declaration

Syntax

Var *array_name* **: Array [***lower_bound* **..** *upper_bound***]** *elements_type;*

or

Type *array_name* **= Array [***lower_bound* **..** *upper_bound***]** *elements_type;*

✓ **Examples :**

```
Type    Average_array = Array [1..20] of real;
        names_array= Array [1..100] of string;

Var     averages : Average_array ;
        students_name, profs_name : names_array;
        Marks : Array [1..100] of real;
        Coefficient : Array [1..20] of integer;
```

Manipulation

✓ **Access** to an array element is done by ***specifying the name of the array*** followed by the value of the ***index*** in square brackets.

Syntax : reading / consultation

Variable_name ← *Array_name* **[***element_index***]**;

✓ **Examples :**

N1 ← Marks[3];
coef_math ← Coef[i];
AVG_a ← Coef[x1+x2 MOD a];
Marks_alg ← Marks[x]*Coef[y];

Syntax : writing / modification

Array_name **[***element_index***]**← *expression*

✓ **Examples :**

Marks[5] ← N2;
Coef[j] ← coef_phy
Coef[x1*x2 DIV b] ← AVG_b

1D Arrays : Vectors

Declaration

Algorithm

Syntax:

VAR var_name : Array[1.. Long_max] of type ;

C

Syntaxe:

type var_name[long_max]

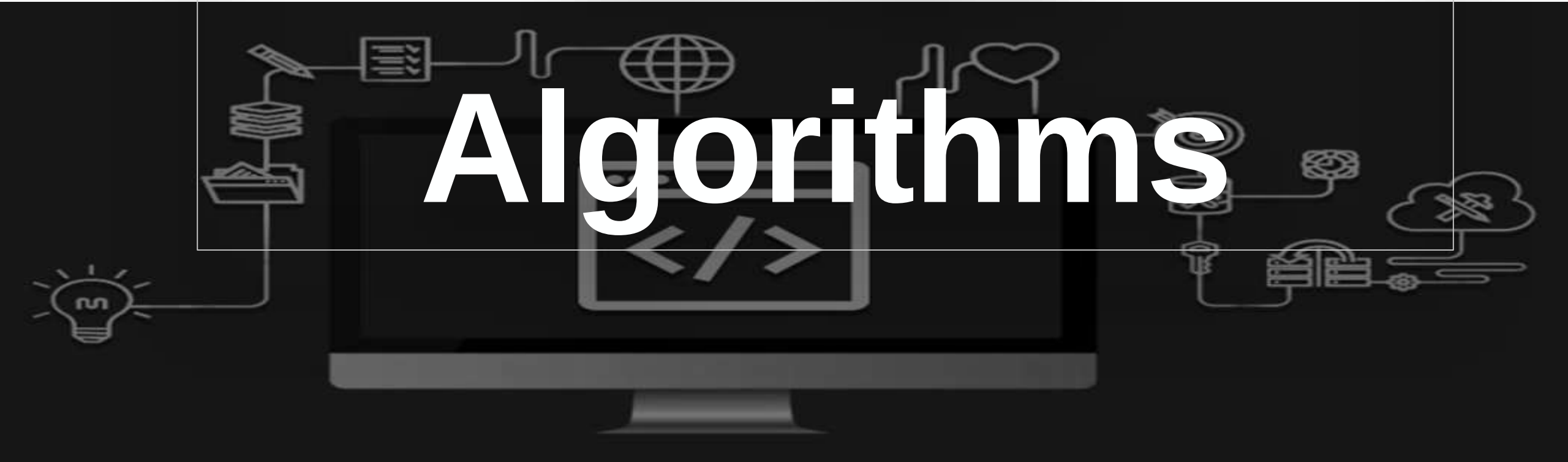
Examples

```
Algorithm Example;


var
    t : Array[1..5] of Integer;
    somme, i : Integer;

begin
    somme ← 0;
    for i← 1 to 5 do
      Read(t[i]);

    for i← 1 to 5 do
      somme ← somme + t[i];

    Write('la somme = ', somme);

end.
```

```
#include <stdio.h>

int main (){

    int t[5];
    int somme = 0;
    int i;

    for (i = 0; i < 5; ++i){
         printf("%d Valeur :", i);
        scanf("%d", &t[i]);


    for (i = 0; i < 5; ++i)
        somme += t[i];

    printf("%d\n", somme);

    return 0;
```

Arrays

Algorithms



Example 1 : Fill an array

Fill an array

Let T be an array of N integer values with

N < 100. Write an algorithm that fills the array T.

Analysis :

Fill an Array → Read the values and save them in an array.

Read an Array → Read the elements of array **T**

T[i]: the i-th element in the array (cell number i)

Read (T[i]): Read a value -from the keyboard- and save it in cell number "i" in array T

```
Algorithm array_fill;
Type Tab = Array[1..100] of integer;
Var T : Tab ;
    N, i : integer;
Begin
   Write ("Give the number of elements of T");
   Read (N);

   For i ← 1 to N Do
      Read(T[i]);

   Endfor

End.
```

Example 2 : Display an array

Display an array

Let T be an array of N integer values with N < 100. Write an algorithm that displays the content of array T.

Analysis :

Display an array → display the content of the array

Display the content of an array → display the value of each cell in the array

Write (T[i]): display -on the screen- the value of cell number "**i**" in array **T**

```
Algorithm array_display;

Var  T : Array[1..100] of integer;
     i : integer;
Begin
   Write (" Give the number of elements of T ");
   Read (N);

     For i ← 1 to N Do
        Write(T[i]);

     Endfor

End.
```

Example 3 : The sum of the elements of an array

The sum of the elements

Let T be an array of N integer values with N < 100. Write an algorithm that calculates and displays the sum of the elements of the array.

Analysis :

Declare a variable Sum **S** and initialize it to zero

Traverse all elements of the array
Each time, add the value of the current cell to the sum **S**

```
Algorithm array_sum;
Type Tab = Array[1..100] of integer;
Var T : Tab;
    S, i : integer;
Begin
   Write (" Give the number of elements of T ");
   Read (N);
   S ← 0;
    For i ← 1 to N Do
        S ← S + T[i];
    Endfor
   Write ("The sum = ", S);

End.
```

2D Arrays:

Matrix

Definition

- ✓ A **two-dimensional** array, also called a **matrix**, is a data structure for organizing information of the same type into **rows** and **columns**.
- ✓ The **size of a matrix** is its ***number of rows*** and ***number of columns***.
- ✓ **Examples**: Matrix “A” is a 2D Array with 4 **rows** and 5 **columns**

A	1	2	3	4	5
1	4	12	8	-4	0
2	15	-2	-7	9	5
3	3	23	39	-10	-1
4	-14	18	5	2	-19

Declaration

Syntax

Var *Array_name* **: Array [***1***..** *nb_rows***,** *1* **..** *nb_columns***]** *Type_elements;*

Or

Type *Array_Name* **= Array [***1***..** *nb_rows***,** *1* **..** *nb_columns***]** *Type_elements;*

✓ **Examples :**

```
Type    Array_Average = Array [1..20, 1..50] of real;
        Array_Name = Array [1..100, 1..100] of String;

Var     Average : Array_Average ;
        Student_name, Prof_name : Array_Name;
        Marks : Array [1..100, 1..100] of real;
        coefficient : Array [1..20, 1..20] of integer;
```

Manipulation

✓ **Access** to an Array element is carried out by specifying the ***Array name*** followed by the value of the ***row index*** and the ***column index*** in square brackets (separated by **,**).

Syntax : reading / consultation

Variable_name←*Array_name*[*row_index,column_index*];

✓ **Examples :**

N1 ← Marks[3,4];
coef_math ← Coef[i,j];
moy_a ← Coef[x1+x2, b MOD a];
Marks_alg ← Marks[x1,y1]*Coef[x2,y2];

Syntax : writing / modification

Array_name [*row_index, column_index*]← *expression;*

✓ **Examples :**

Marks[5,2] ← N2;
Coef[j,k] ← coef_phy
Coef[x1*x2, a DIV b] ← AVerage_b

2D Arrays : Matrix « reading and writing »

Declaration

Algorithm

Syntax:

VAR var_name**= Array**[1.. Max_row, Max_column] **of** type ;

C

Syntaxe:

type nom_var**[**Max_ligne**][**Max_col**]**

Examples

```
Algorithm  Example;
var
    m : Array[1..5,1..10] of Integer;
    i, j : Integer;

begin
    somme  ←  0;
    for i ←  1 to 5 do
       for j ←  1 to 10 do
         Read(m[i,j]);

    ...

end.
```

```
#include <stdio.h>

int main (){

    int m[5][10];
    int i,j;

    for (i = 0; i < 5; ++i)
       for (j = 0; i < 10; ++i)
          scanf("%d\n", m[i][j]);

    ...

    return 0;
}
```

Algorithms

Dealing with Matrices

Example 1 : fill an matrix

Fill an Array

Let A be a two-dimensional array of N integer values with N < 100. Write an algorithm that fills the array A.

Analysis :

Fill an array → Read values and save them in an array

T[I,j]: the i-th and j-th element in the array (cell indexed by i and j)

Read (T[i,j]): Read a value -from the keyboard- and save it in cell indexed by i, j in array T

```
Algorithm  Example;
var
    m : Array[1..5,1..10] of Integer;
    i, j : Integer;

begin
    somme  ←  0;
    for i ←  1 to 5 do
       for j ←  1 to 10 do
         Read(m[i,j]);
        endfor;
    Endfor;

    ...

end.
```

Example 2 : display a matrix

Remplir un Array

Let A be a two-dimensional array of N integer values with N < 100. Write an algorithm that displays the array A.

Analysis :

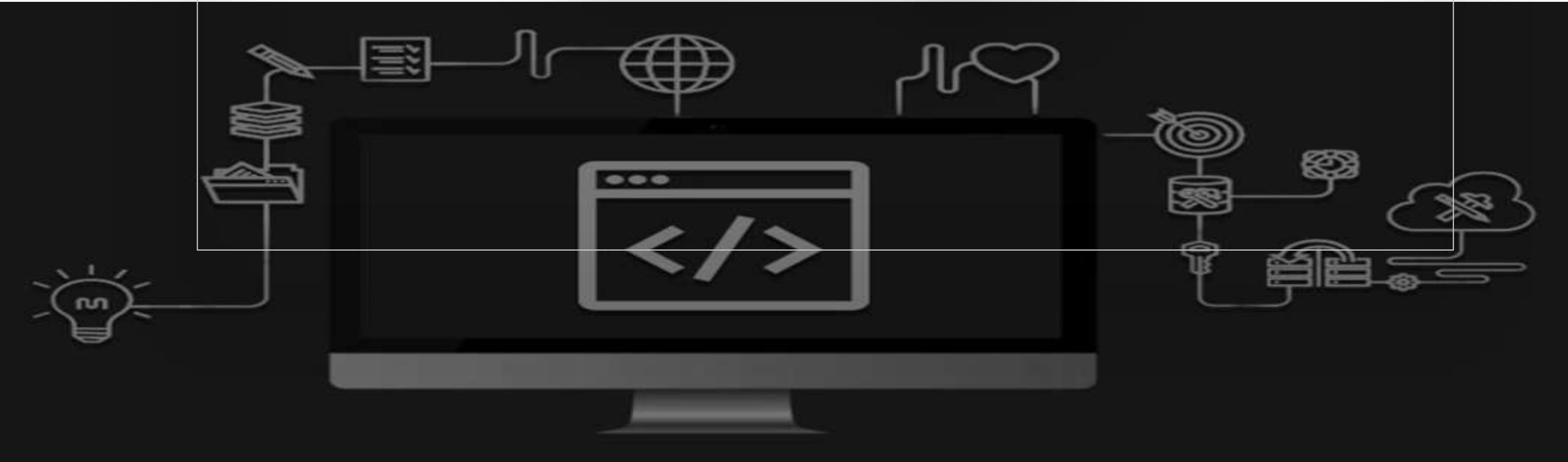
Display an array → display the content of the array

Display the content of an array → display the value of each cell in the array

Write (T[I,j]): display -on the screen- the value of cell number I , j in array T

```
Algorithm  Example;
var
    m : Array[1..5,1..10] of Integer;
    i, j : Integer;

begin
    somme  ←  0;
    for i ←  1 to 5 do
       for j ←  1 to 10 do
         write(m[i,j]);
        endfor;
    endfor;

    ...

end.
```

Strings



Definition

- ✓ A string is a **one-dimensional array** of characters that can be manipulated **as an array**

Syntax : Algorithm

name_string : array [1..25] of char;

Syntax : in C

Char name_string [25]; **//In C programming, a string is actually an array of characters terminated by a null character ('\0')**

Declaration in C programming

```
// Method 1: Character array

char str1 [20];

// Method 2: String literal

char str2[] = "Hello";

// Method 3: With explicit size

char str3[6] = "World";
```

2. String Initialization

```
#include <stdio.h>
int main() {
    // Different ways to initialize strings
    char str1[] = "Hello"; // Automatic null termination
    char str2[6] = {'W', 'o', 'r', 'l', 'd', '\0'};
    char str3[10];
     printf("str1: %s\n", str1);
    printf("str2: %s\n", str2);
    return 0;}
```

3. Important String Functions from <string.h>

A. strlen() - String Length

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Programming";
    int length = strlen(str);
        printf("String: %s\n", str);
    printf("Length: %d\n", length); // Output: 11
        return 0;}
```

B. strcpy() - String Copy

```
#include <stdio.h>
#include <string.h>
int main() {
    char source[] = "Copy this";
    char destination[20];
     strcpy(destination, source);
    printf("Source: %s\n", source);
    printf("Destination: %s\n", destination);
      return 0;}
```

C. strcat() - String Concatenation

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20] = "Hello ";
    char str2[] = "World!";
    strcat(str1, str2);
    printf("Concatenated: %s\n", str1); // Output: Hello World!
    return 0;}
```

D. strcmp() - String Comparison

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "apple";
    char str2[] = "banana";
    int result = strcmp(str1, str2);
     if(result == 0)     printf("Strings are equal\n");
    else if(result < 0)   printf("'%s' comes before '%s'\n", str1, str2);
    else printf("'%s' comes after '%s'\n", str1, str2);
     return 0;
```

4. Algorithm Examples with Strings

Algorithm 1: Count Vowels in a String

```
#include <stdio.h>

#include <ctype.h>

int countVowels(char str[]) {

    int count = 0;

    int i = 0;

     while(str[i] != '\0') {        char ch = tolower(str[i]);  if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 
'u') {   count++;     }   i++;  }  return count;}

int main() {    char text[100];     printf("Enter a string: ");    fgets(text, sizeof(text), stdin);

    int vowels = countVowels(text);

    printf("Number of vowels: %d\n", vowels);

     return 0;}
```

4. Algorithm Examples with Strings

Algorithm 2: Reverse a String

```
#include <stdio.h>
#include <string.h>
void reverseString(char str[]) {   int length = strlen(str);   int start = 0;  int end = length - 1;
      while(start < end) {
        // Swap characters
        char temp = str[start];       str[start] = str[end];     str[end] = temp;      start++;    end--;   }}
int main() {
    char str[100];   printf("Enter a string: ");  fgets(str, sizeof(str), stdin);
        // Remove newline character
    str[strcspn(str, "\n")] = 0;
     printf("Original: %s\n", str);
    reverseString(str);
    printf("Reversed: %s\n", str);    return 0;}
```

To Do ?????

Algorithm 3: Check if String is Palindrome
Algorithm 4: Count Words in a String
Algorithm 5: Find Most Frequent Character

Ministry of Higher Education and Scientific Research
Djilali BOUNAAMA University - Khemis Miliana(UDBKM)
Faculty of Matter Science and Computer Science
Department of Mathematics

Chapter : 6

Arrays and Strings

MI-L1-UEF121 : Algorithms and Data Structures I

Ali Khalfi

Khalfiali.udbkm@gmail.com