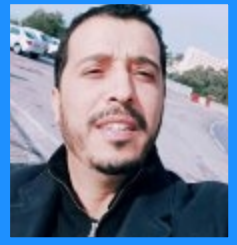
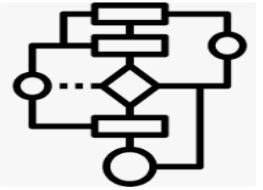
Algorithms and Data Structure 01



Dr. Sabri Ghazi
sabri.ghazi@univ-annaba.dz
Computer Science Department
Badji Mokhtar Annaba University

Chapter 05 Arrays



See this algorithm and tell what is doing

```c
#include<stdio.h>
int main(){
    float exam=13.5;
    float tutorial=5;
    float practical=3.5;

    float cc=tutorial+practical;

    float final_grad=(exam*3+cc)/4;
    printf("\n Your final grad is %f \n",final_grad);

}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL cppdbg: Array_demo_1

```
Your final grad is 12.250000
```

Definition : Array

An **array** is a **data structure** that stores a **collection of elements**, each identified by an **index** or a key. The elements are stored in **contiguous memory locations**, and the **index** serves as a reference to access a specific element

Properties of Array

Fixed Size: The number of elements they can store is determined at the time of declaration. Once the size is set, it typically cannot be changed during runtime.
Homogeneous Elements: All elements in an array must be of the same data type. For example, an array can store integers, floating-point numbers, characters, etc., but all elements within a particular array must be of the same type.
Contiguous Memory Allocation: Array elements are stored in adjacent memory locations. This allows for efficient access to elements using their indices because the location of each element can be calculated based on the index and the size of the elements.
Zero-Based Indexing: In many programming languages, array indices start from 0. This means the first element is accessed using index 0, the second with index 1, and so on.

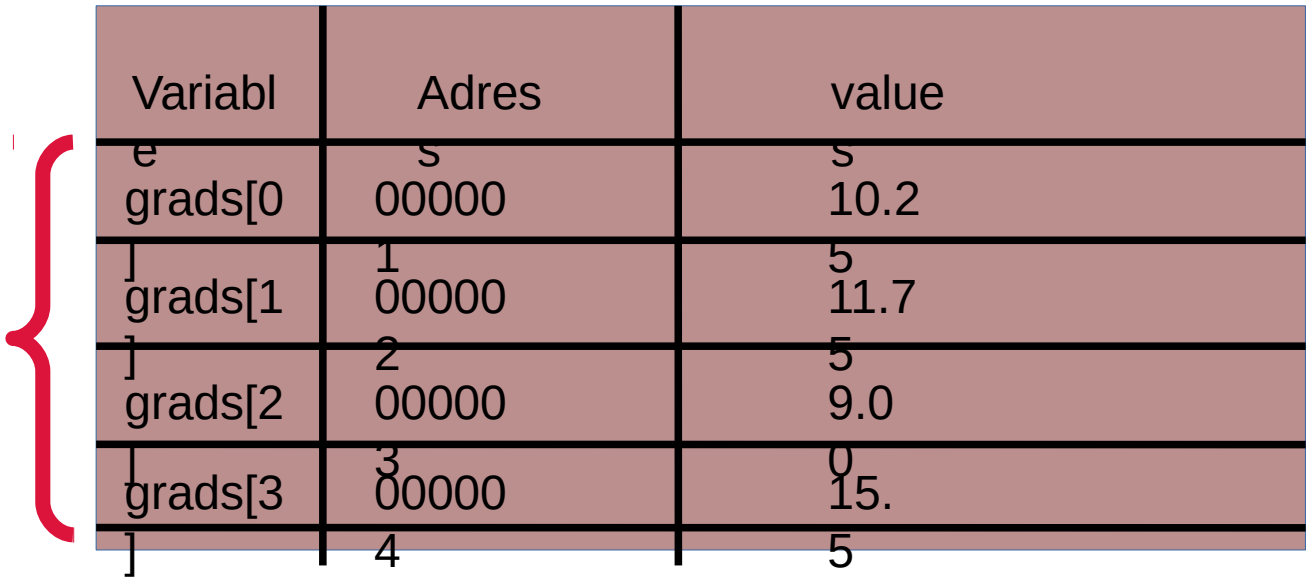
Arrays

- The algorithm compute the final grad of **one student**.
- How can we modify it and adapt it to compute the final grads for **all the students** ?
  - We have **23 students**
  - Should we declare **23** variables for exams and **23** variable for the practical and ….
- → Is there any other solution that help us manipulate all the information in one single variables ?

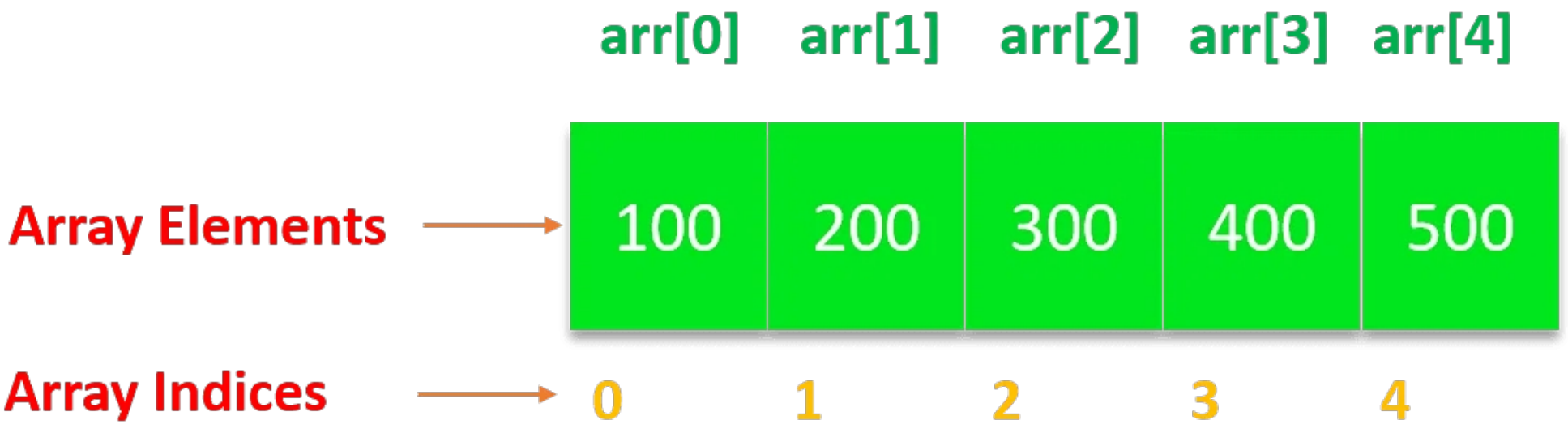
Arrays

- As illustrated in the figure, an array is a space in the memory declared to help manipulate many information which have the same type!

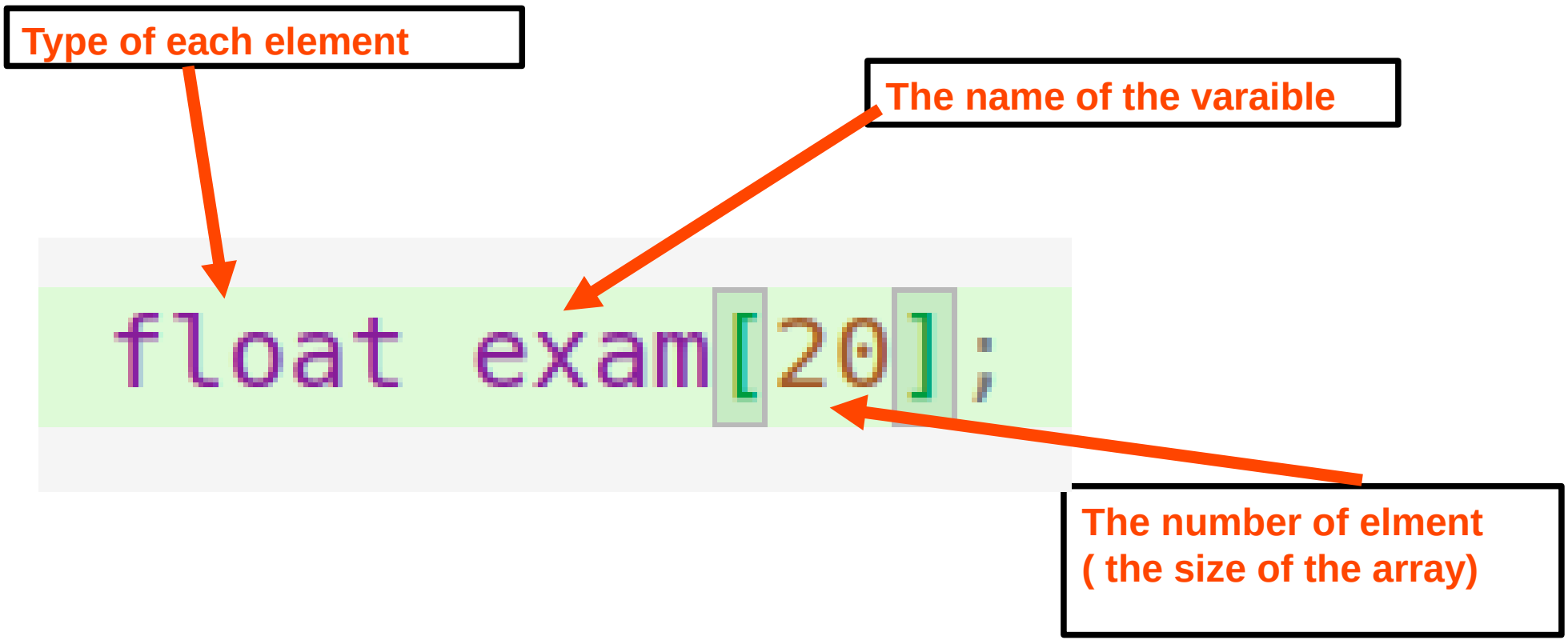
float grads[4];



Arrays


arr[0]
arr[1]
arr[2]
arr[3]
arr[4]
Array Elements
100
200
300
400
500
Array Indices
0
1
2
3
4

Syntax to declare an Array



Syntax to declare an Array and initialize it

Type of each element

The values, must be of same type and also with the same number here 5 elements

```
3       int arr[5]={10, 12, 15, 10, 13};
```

Syntax to acess to a cell of an array

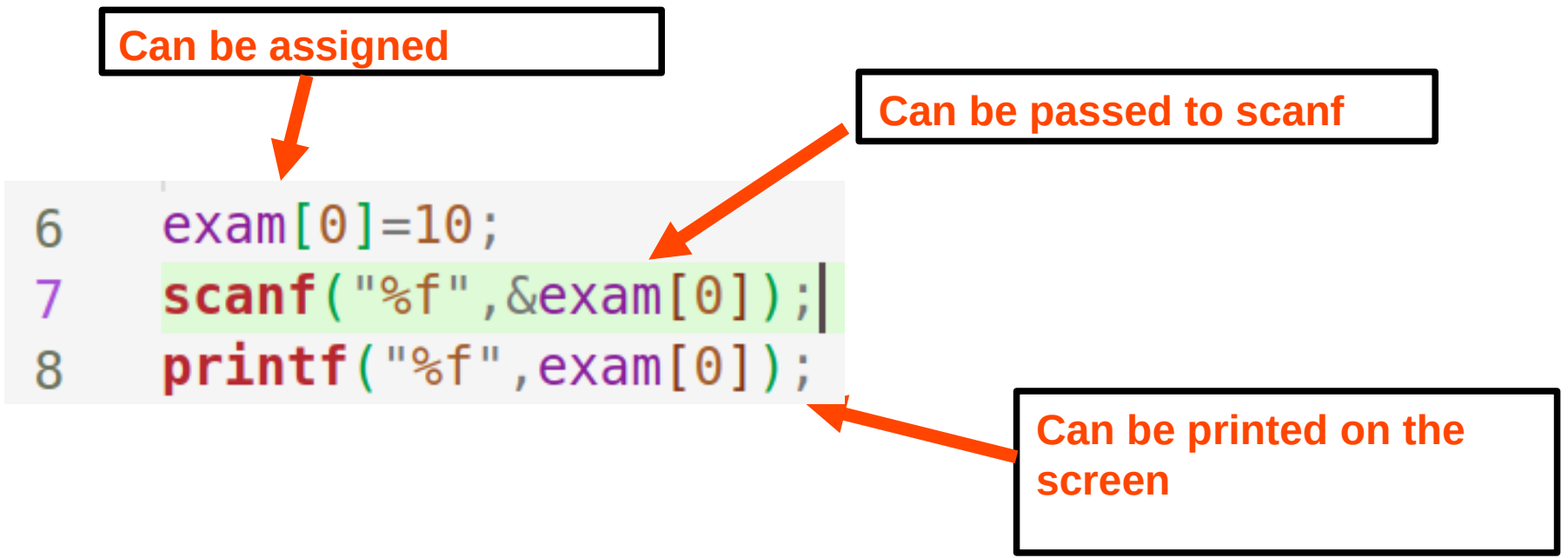
```
int arr[100];

arr[20]=12.25;
```

The array

The indice of the element
Always < the size of the arra

Syntax to acces to an element in the array



Syntax to acces to an element in the array

Once declared an array we could apply the following operations :

1) **Display** the array.
2) Let the user **introduce the values** of each element.
3) **Find** if the array **contains** or not a specific element.
4) **Count** the number of occurrence of a specific element in the array.
5) Find the maximum finding the minimum, etc

Declare an array and display it

```c
#include<stdio.h>
int main(){
    int arr[5]={10, 12, 15, 10, 13};
    int i;
    for(i=0;i<5;i++){
        printf("Index %d value %d \n ",i,arr[i]);
    }
}
```

We need a loop in order to Acess all the elements of an array

Introduce the elements of the the Array :

```
1   #include<stdio.h>
2   int main(){
3   int arr[10];
4   int i;
5   for (i=0; i<10;i++)
6   {
7       printf("Enter number %d \n", i);
8       scanf("%d", &arr[i]);
9   }
10  }
```

Allow the user to input the value of the element.

Arrays are important in Algorithms !

Arrays excel at organizing data in a contiguous memory block, allowing for easy indexing and access to individual elements.

- One of the standout features of arrays is their constant-time random access. Retrieving an element based on its index takes the same amount of time, irrespective of the array size.

- Efficient Searching and Sorting: Arrays facilitate efficient search and sort operations, contributing to algorithmic performance

Demo #1 declare an array and display it

Write an algorithm that ask the user to intoduce a serie of 5 numbers and then compute their sum, average.

Can you modify it to find the maximum value ?

Demo #2 count the number of odd elements

Write an algorithm, that given an array of 10 values, it count the number of odd numbers.

Multidimensional Arrays

Arrays are very usefull tool to store data of the same type, however when we want to store information in tabular form we nee a two dimensions: with rows and columns.
In this case we need **Two-Dimensional** arrays also called **Matrix**.

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

```
int matrix[2][3]={{1,4,2},{3,6,8}};
```

Access the Elements of a 2D Array

To access an element of a two-dimensional array, you must specify the index number of both the row and column.

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

```
int matrix[2][3]={{1,4,2},{3,6,8}};
printf("%d",matrix[1][2]);
```

The indice of the row

The indice of the column

How a 2D Array is really represented in memory ?

RUN AND DEBUG ···

VARIABLES

Locals

matrix: [2]

[0]

[0]: 1

[1]: 4

[2]: 2

[1]

[0]: 3

[1]: 6

[2]: 8

Array_demo_1.c | Array_demo_2.c | Array_demo_3.c

home > user > Desktop > ALGO-01-Hybrid-Computing-Automation > Demostrat

```c
#include<stdio.h>
int main(){

int matrix[2][3]={{1,4,2},{3,6,8}};
printf("\n %d \n",matrix[1][2]);

}
```

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Access the Elements of a 2D Array

To access an element of a two-dimensional array, you must specify the index number of both the row and column.

- **Row index must be < then the number of rows**
- **Column index must be < then the number of columns**

Operations we could do on matrix

- Matrix Addition
- Matrix Subtraction
- Matrix Multiplication
- Scalar Multiplication
- Transpose of a Matrix
- Diagonal
- etc

Operations we could do on matrix

```c
#include<stdio.h>
int main(){
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
int i, j;
for (i = 0; i < 2; i++) {
  for (j = 0; j < 3; j++) {
    printf("%d\n", matrix[i][j]);
  }
}
}
```

We need two nested loops to be able to visit all the cells of the matrix.
- the first for the row index
- the second for the column index