Array in C

28 February 2026 20:18

I want to store marks of 50 students of a class. And, I need to find out, (marks: numerical data)

i.) what is the highest marks and its roll_number, (roll_number: numerical data.)

ii) what is the lowest marks and its roll_number,

iii) what is the average marks of the class.

iv) List all the roll_numbers have got above average and below average.

Array is a linear data structure that stores multiple values of the same data type in contiguous memory locations, accessed by using index values.

In the above problem, without array we need to declare multiple variables. If we declare multiple variables the problem which we face:-

⇒ Hard to manage.

⇒ No iteration abstraction.

⇒ No algorithmic processing (search, sum, etc.).

By using array the whole will become very simple.

```
float marks[50];
int roll_number[50];
```

One variable → multiple values → loop processing possible

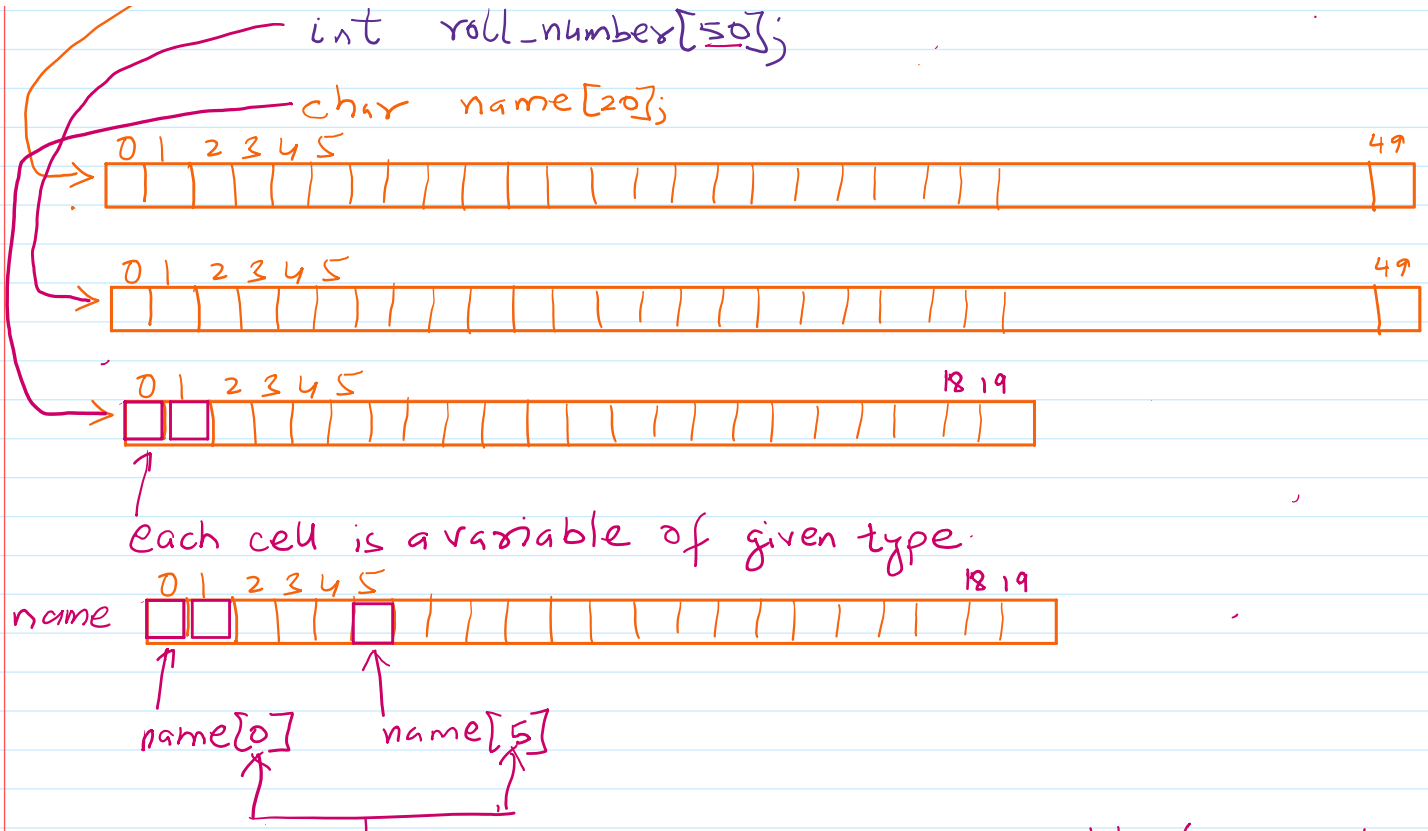
Syntax:

```
data-type array_name[size];
```

Note: starting index is '0' zero. Hence, end inde is always 'size − 1'.

Example:

```
float marks[50];
int roll_number[50];
```



Initialization of 1D Array.

```
int num[5] = {10, 20, 30, 40, 50};   // method-1
int n[] = {100, 200, 300, 400};      // method-2.
```

size is not mentioned, it is defined by the compiler.

Compiler calculates the size = 4.

```
int num[5] = {10, 20};   // method-3  partial initialization.
```

Memory → | 0 | 1 | 2 | 3 | 4 |
| --- | --- | --- | --- | --- |
| 10 | 20 | 0 | 0 | 0 |

Accessing elements

```
array-name[index];
```

rule 1: 'index' cannot be -ve.
rule 2: 'index' is always less than 'size'.

rule 1: 'index' cannot be -ve.
rule 2: 'index' is always less than 'size'
otherwise, we get runtime error.

Example:

```
int num[5] = {10, 20, 30, 40, 50};
printf("%d", num[2]);   // output: 30
num[4] = 100;   // it will modify the 5th index value.
```

Memory →

0	1	2	3	4
10	20	30	40	100

C-language is the fastest programming language

Input and Output using Loop:

preprocessor directive
standard folder in C-language
header file, it contains compiled code
store this constant in the processor.

```c
#include<stdio.h>
#define asize 5
int main(int argc, char const *argv[])
{
    int num[asize], i;
    printf("Enter %d numbers:\n",asize);
    for(i = 0; i < asize; i++){
        scanf("%d", &num[i]);
    }
    printf("\nArray elements are: [");
    for(i = 0; i < asize - 1; i++){
        printf("%d, ",num[i]);
    }
    printf("%d]\n",num[i]);
    return 0;
}
```

Output:

```
Enter 5 numbers:
77
61
91
10
26

Array elements are: [77, 61, 91, 10, 26]
```

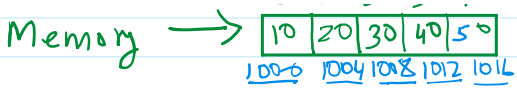
Memory Representation(Core Concept)
Arrays use **contiguous memory allocation.**

```
int num[5] = {10, 20, 30, 40, 50};
```

size - 4 bytes

Memory →

0	1	2	3	4
10	20	30	40	50
1000	1004	1008	1012	1016



* base address = 1000
* size of int is 4 bytes

Address formula:

address of num[i] = base_address + (i * size_of_data_type)

```c
#include<stdio.h>
#define asize 5
int main(int argc, char const *argv[])
{
    int num[asize], i;
    printf("Enter %d numbers:\n",asize);
    for(i = 0; i < asize; i++){
        scanf("%d", &num[i]);
    }
    printf("\nArray elements are: [");
    for(i = 0; i < asize - 1; i++){
        printf("%d, ",num[i]);
    }
    printf("%d]\n",num[i]);
    printf("\nMemory address of each element: [");
    for(i = 0; i < asize - 1; i++){
        printf("%u, ",&num[i]);
    }
    printf("%u]\n",&num[i]);
    printf("\nBase address: %u",num);
    return 0;
}
```

↑ name of array.

Output:

```
Enter 5 numbers:
2
5
1
3
6

Array elements are: [2, 5, 1, 3, 6]

Memory address of each element: [6291120, 6291124, 6291128, 6291132, 6291136]

Base address: 6291120
```

Note: The name of the array stores the base address.

* The name of array itself is a constant pointer.

pointer is a variable, which keeps the address or reference of another variable as a content.

↑ content of num.

This is the proof that array elements are stored in contiguous memory allocation.

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

← rand()

```c
#include<stdio.h>
#include <stdlib.h>
#include <time.h>
#define asize 50
int main(int argc, char const *argv[])
{
    srand(time(NULL)); // Seed with current time
    float marks[asize];
    int roll_num[asize], i;
    int max_marks_index, min_marks_index;
    for(i = 0; i < asize; i++){
        marks[i] = rand() %101 ;
        roll_num[i] = i+101;
    }
    printf("\nRoll number\tMarks\n");
    for(i = 0; i < asize ; i++){
        printf("%d\t\t%.0f\n",roll_num[i],marks[i]);
    }
    return 0;
}
```

rand()

time(NULL)

← this randomizes by using system time, such that every execution will give you different random values, when you call rand() function.

H.W :- do the rest part of the question.

Output:

```
Roll number     Marks
101             41.0
102             51.0
103             23.0
104             50.0
105             10.0
106             97.0
107             77.0
108             21.0
109             44.0
110             24.0
111             48.0
112             39.0
113             75.0
114             30.0
115             6.0
116             45.0
117             73.0
118             75.0
119             53.0
120             88.0
121             86.0
122             94.0
123             42.0
124             10.0
125             74.0
126             69.0
127             72.0
128             11.0
129             95.0
130             74.0
```

131	73.0
132	2.0
133	74.0
134	46.0
135	82.0
136	65.0
137	85.0
138	84.0
139	49.0
140	33.0
141	12.0
142	23.0
143	43.0
144	0.0
145	16.0
146	19.0
147	58.0
148	7.0
149	69.0
150	23.0