Introduction to Programming in C

(IS-FEE-10061S)

Białystok University of Technology
Faculty of Electrical Engineering
Academic year 2023/2024

Workshop no. 05 (28.03.2024)

Jarosław Forenc, PhD

Topics

- One-dimensional arrays in C
  - declaration
  - access to array elements
  - element initialization
  - pseudo-random number generator
  - operations

Arrays in C

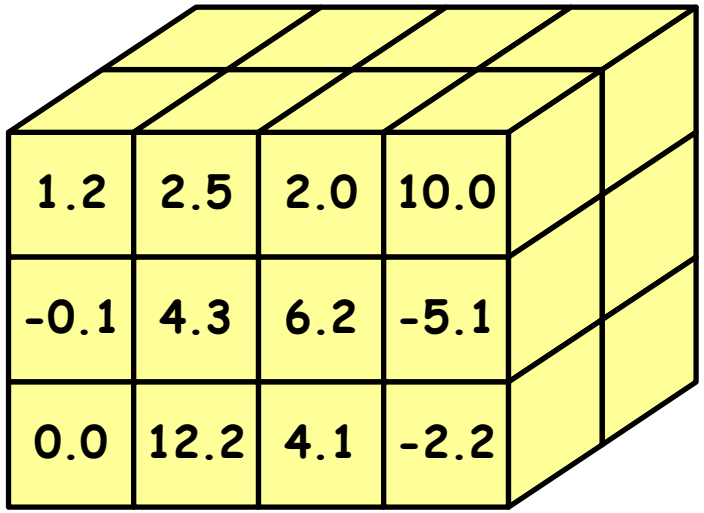
- Array - a series of values of the same type (elements), stored sequentially

vector

5	3	-2	1	-4

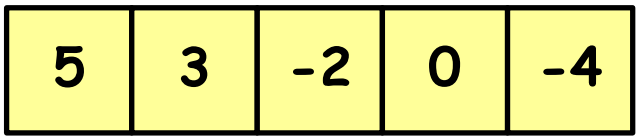
matrix

a	c	d	m
p	d	q	l
a	t	x	v

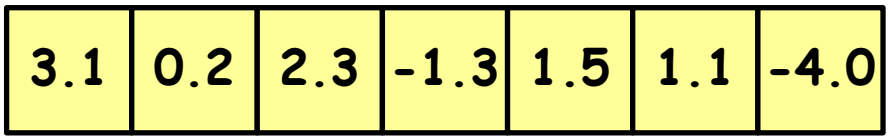


One-dimensional array

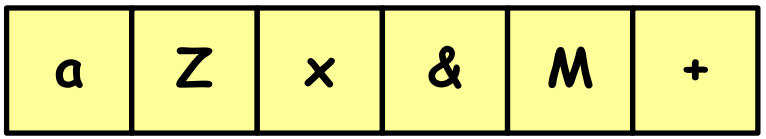
- Array - a series of values of the same type (elements), stored sequentially
- Vector - one-dimensional array



- integers

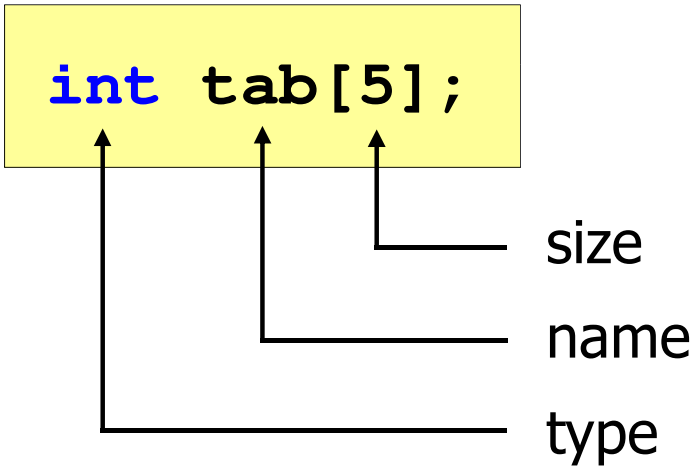


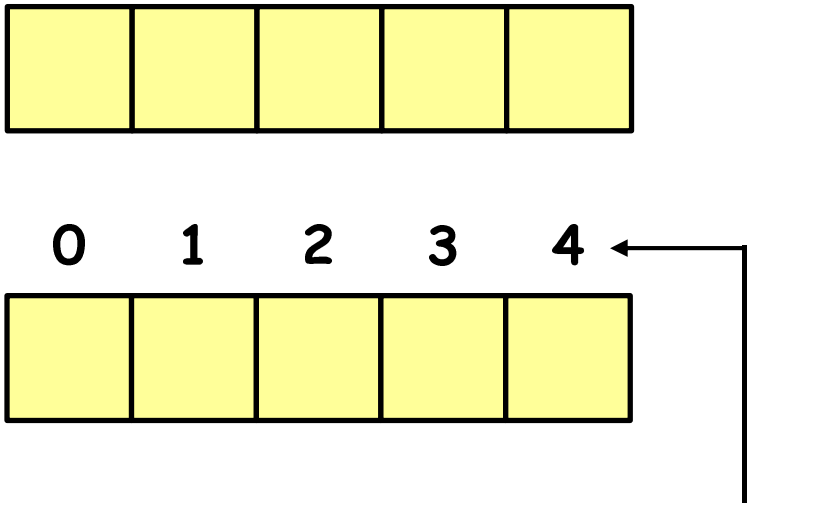
- real numbers



- characters

One-dimensional array: declaration





- Array size is the value:
  - integer, positive
  - known at the compilation stage
    (number: 5, #define N 5, const int n = 5;)

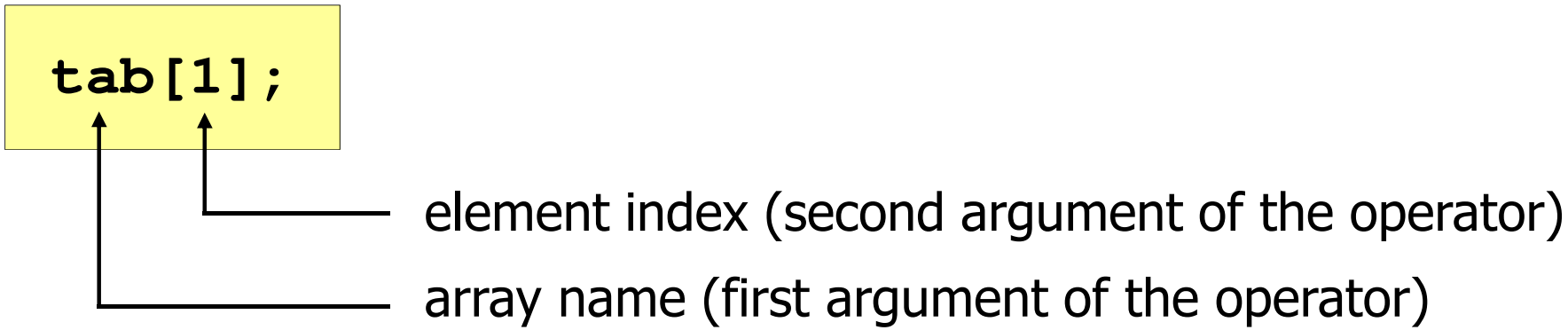
```
int tab[5];
```

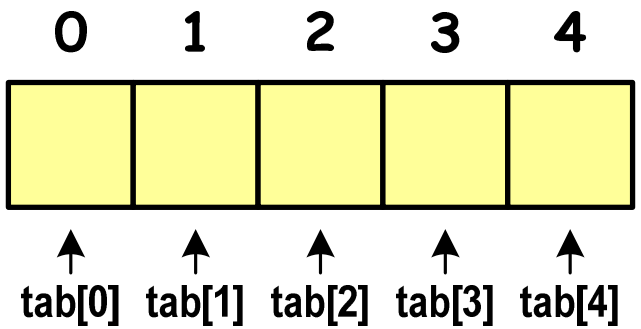
```
int tab[N];
```

```
int tab[n];
```

One-dimensional array: access to array elements

[ ] - indexing operator

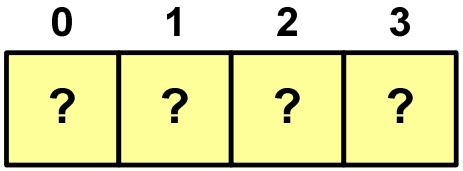




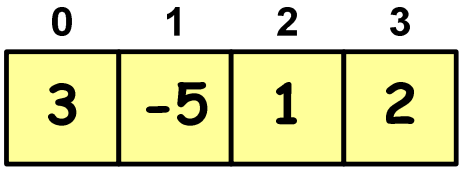
- Index:
  - number, e.g. 0, 1, 10
  - variable, e.g. i, idx
  - expression, e.g. i*j+5

One-dimensional array: access to array elements

```
int tab[4];
```




```
tab[0] = 3;
tab[1] = -5;
tab[2] = 1;
tab[3] = 2;
```



- Each element of the array is treated in the same way as an int variable

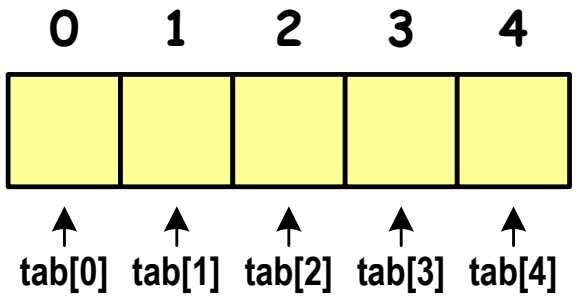
```
printf("%d",tab[0]);
```

```
scanf("%d",&tab[1]);
```

One-dimensional array: access to array elements

- The compiler doesn't check to see whether the indices are valid

```
int tab[5];
tab[5] = 10;
```



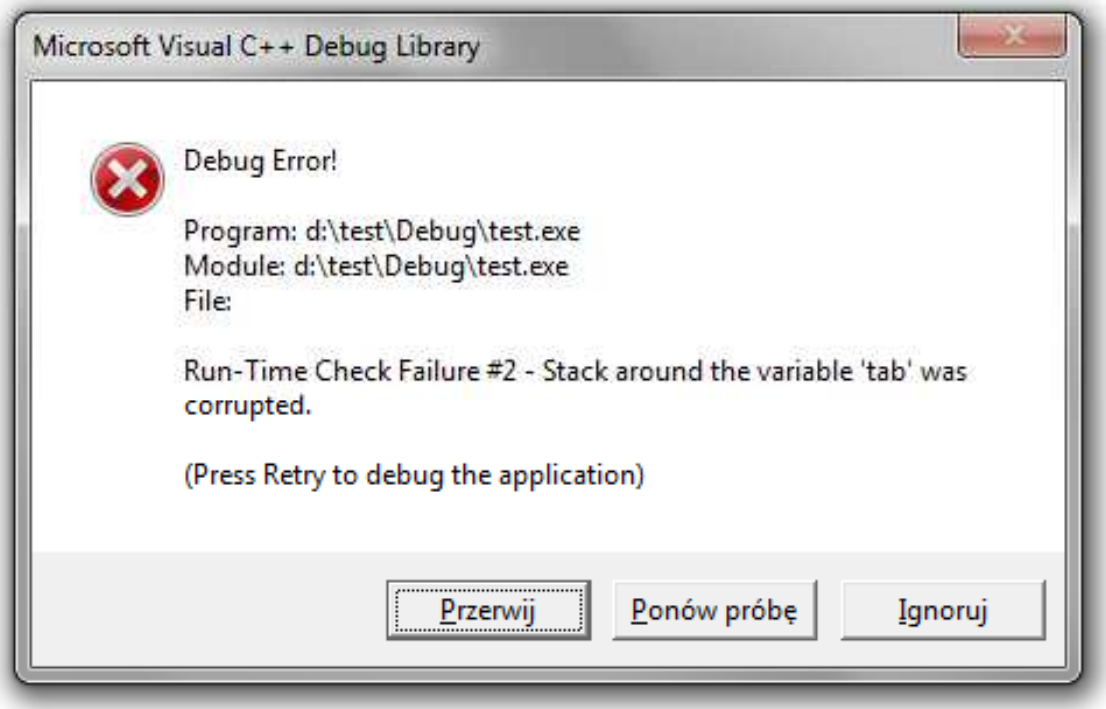
- error!!! - tab[5] element does not exist

- The compiler will not signal an error
- The program will execute the operation
- IDE may indicate a problem

One-dimensional array: access to array elements

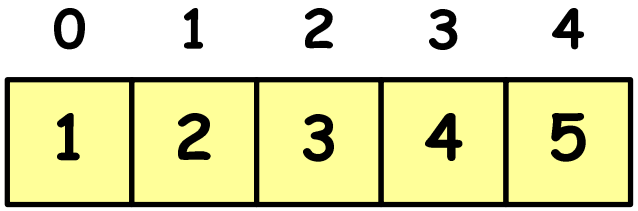
- The compiler doesn't check to see whether the indices are valid

```
int tab[5];

tab[5] = 10;
```

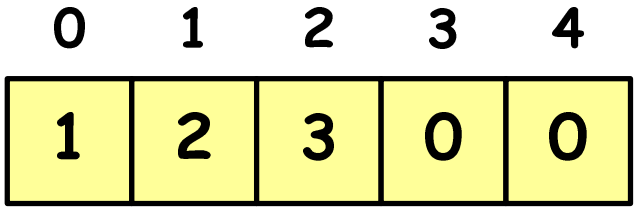


One-dimensional array: element initialization

```
int tab[5] = {1,2,3,4,5};
```



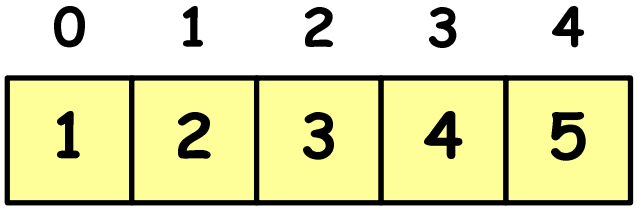

```
int tab[5] = {1,2,3};
```




```
int tab[5] = {1,2,3,4,5,6};
```

- compilation error

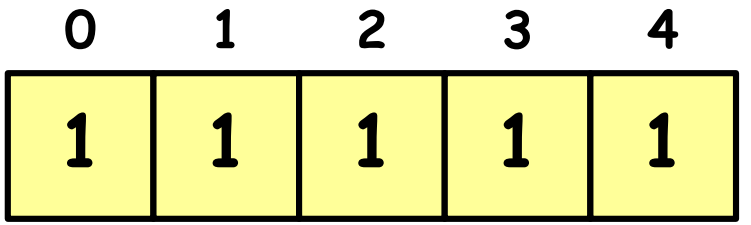
```
int tab[] = {1,2,3,4,5};
```



One-dimensional array: access to array elements

- Assigning the value of 1 to all elements of the array

```
int tab[5];

tab[0] = 1;
tab[1] = 1;
tab[2] = 1;
tab[3] = 1;
tab[4] = 1;
```

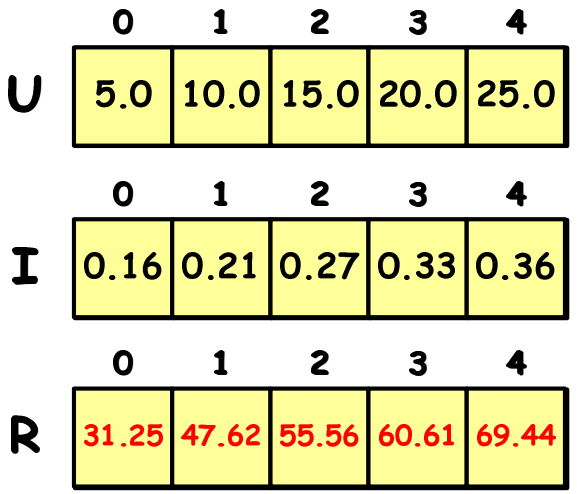



```
int tab[5], i;

for (i=0; i<5; i++)
   tab[i] = 1;
```

Example: operations on large amounts of data

```
R1 = 31.250000
R2 = 47.619048
R3 = 55.555556
R4 = 60.606061
R5 = 69.444444
```

```c
#include <stdio.h>

int main(void)
{
    double U[5] = { 5.0, 10.0, 15.0, 20.0, 25.0 };
    double I[5] = { 0.16, 0.21, 0.27, 0.33, 0.36 };
    double R[5];
    int i;

    for (i=0; i<5; i++)
        R[i] = U[i]/I[i];

    for (i=0; i<5; i++)
        printf("R%d = %f\n",i+1,R[i]);

    return 0;
}
```

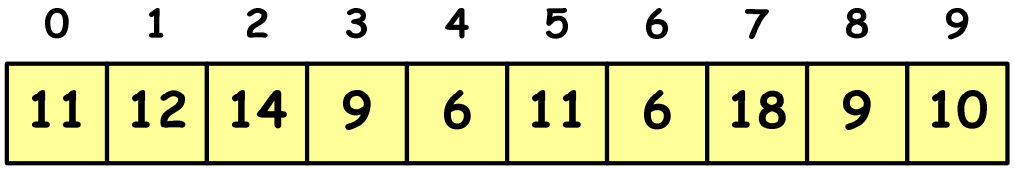


Pseudo-random number generator

- rand() - returns a pseudo-random number within the range of 0 to RAND_MAX (0 … 32767)
- srand() - initializes the pseudo-random number generator
- Header file: stdlib.h (time.h)

```
int x, y, z;
srand((unsigned int) time(NULL));
x = rand();                  // range <0,32767>
y = rand() % 100;            // range <0,99>
z = rand() % (b-a+1)+a;      // range <a,b>
```

One-dimensional array: operations

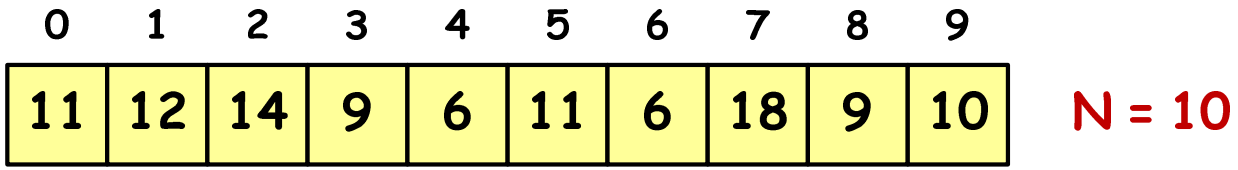



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 10

int main(void)
{
     int tab[N], i;

     /* generating array elements */

     srand((unsigned int) time(NULL));

     for (i=0; i<N; i++)
          tab[i] = rand() % 20;
```

One-dimensional array: operations

```
/* printing array elements */

printf("Array elements:\n");
for (i=0; i<N; i++)
    printf("%d  ",tab[i]);
printf("\n");
```

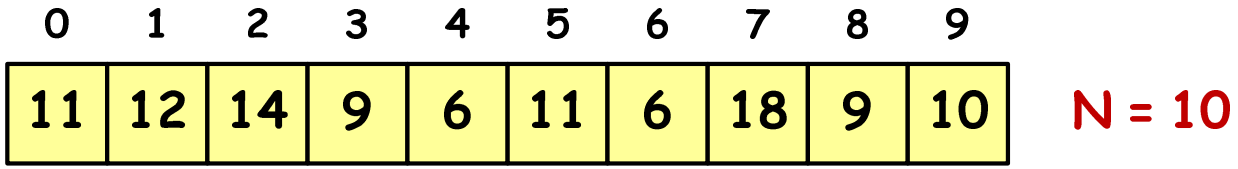
```
Array elements:
11  12  14  9  6  11  6  18  9  10
```



One-dimensional array: operations

```
/* printing elements in reverse order */

printf("Elements in reverse order:\n");
for (i=N-1; i>=0; i--)
    printf("%d  ",tab[i]);
printf("\n");
```

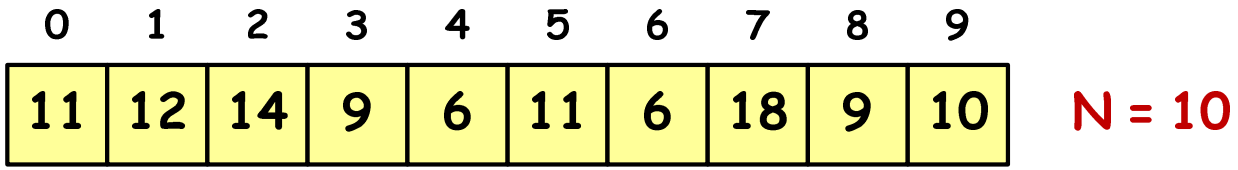
```
Elements in reverse order:
10  9  18  6  11  6  9  14  12  11
```



One-dimensional array: operations

```
/* finding the element with the minimum value */

int min;

min = tab[0];
for (i=1; i<N; i++)
    if (tab[i] < min)
        min = tab[i];
printf("Minimum element value: %d\n",min);
```

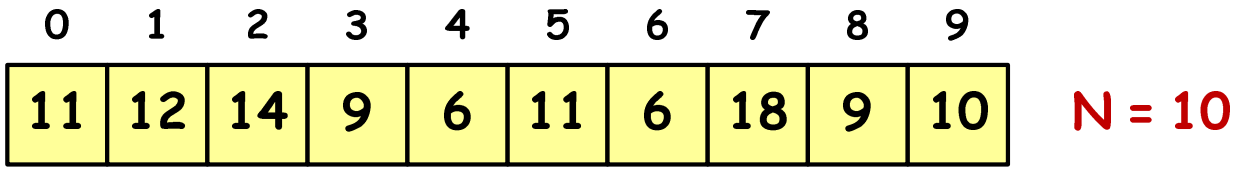
```
Minimum element value: 6
```



One-dimensional array: operations

```
/* finding indices of elements with a minimum value */

printf("Minimum element indices : ");
for (i=0; i<N; i++)
    if (tab[i] == min)
        printf("%d  ",i);
printf("\n");
```

```
Minimum element indices: 4   6
```



One-dimensional array: operations

```
/* sum and arithmetic mean of array elements */

int sum = 0;
float mean;

for (i=0; i<N; i++)
    sum = sum + tab[i];
mean = (float) sum/N;
printf("Sum: %d, mean: %g\n",sum,mean);
```

```
Sum: 106, mean: 10.6
```

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

One-dimensional array: operations

```
/* the number of even elements in the array */

int count = 0;

for (i=0; i<N; i++)
    if (tab[i]%2==0)
        count++;
printf("The number of even elements: %d\n",count);
```

```
The number of even elements: 6
```

0	1	2	3	4	5	6	7	8	9
11	12	14	9	6	11	6	18	9	10

N = 10

End of workshop no. 05

Thank you for your attention!