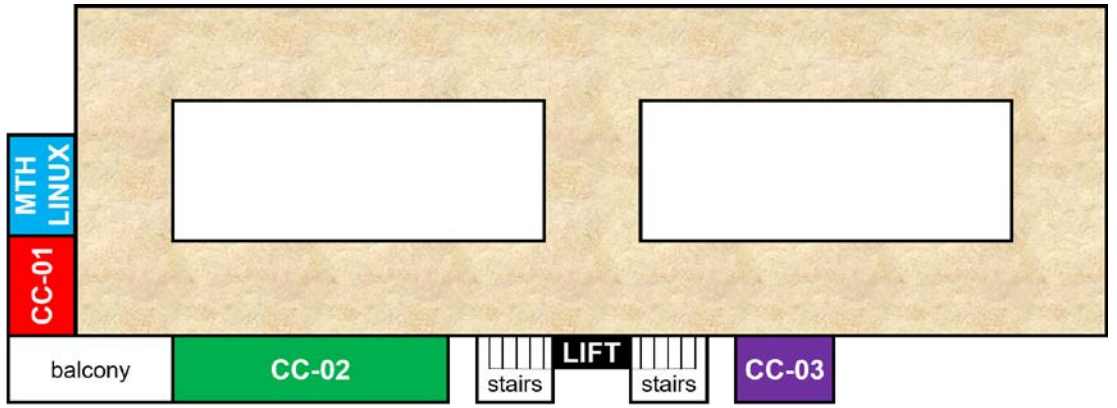
Introduction to Arrays

ESC101: Fundamentals of Computing

Nisheeth

Mid-sem Lab Exam: February 15 (Saturday)

- Morning exam
  - 10:00 AM - 12:30 PM – starts 10:00 AM sharp
  - **CC-01**: A9, {A14 even roll numbers}
  - **CC-02**: A7, A10, A11
  - **CC-03**: A12
  - **MATH-LINUX**: A8, {A14 odd roll numbers}
- Afternoon exam
  - 12:45 PM – 3:15 PM – starts 12:45 PM sharp
  - **CC-01**: A1, {A2 even roll numbers}
  - **CC-02**: A4, A5, A6
  - **CC-03**: A3
  - **MATH-LINUX**: A13, {A2 odd roll numbers}



Mid-sem Lab Exam: February 15 (Saturday)

- Go see your room during this week's lab
- Be there 15 minutes before your exam time
  - No entry for candidates arriving later than 09:45 for morning exam and 12:30 pm for the afternoon exam
- Cannot switch to another session (morning to afternoon or vice-versa)
- Syllabus – till functions (no arrays)
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone or tablet
- **POSSESSING ANY OF THESE WILL BE CONSIDERED CHEATING**
- Prutor CodeBook will be unavailable during lab exam
- Exam will be like labs - marks for passing test cases
- Marks for writing clean indented code, proper variable names, a few comments – illegible code = poor marks

How to store lots of values of same type?

- We have seen many data types so far
  - int, float, char, etc.
- We can use a variable to store a single value of some data type, e.g.,
  - int x = 2; // variable x stores one integer value
  - float x = 2.3; // variable x stores a one float value
  - char x = 'c'; // variable x stores one char value
- What if we want to store several values, all of same data type?
  - Example 1: Marks of all ESC101 students in Major Quiz 1 (all floats)
  - Example 2: Roll numbers of all ESC101 students (all int)
  - Example 3: Final grade of all ESC101 students (all char)

How to store lots of values of same type?

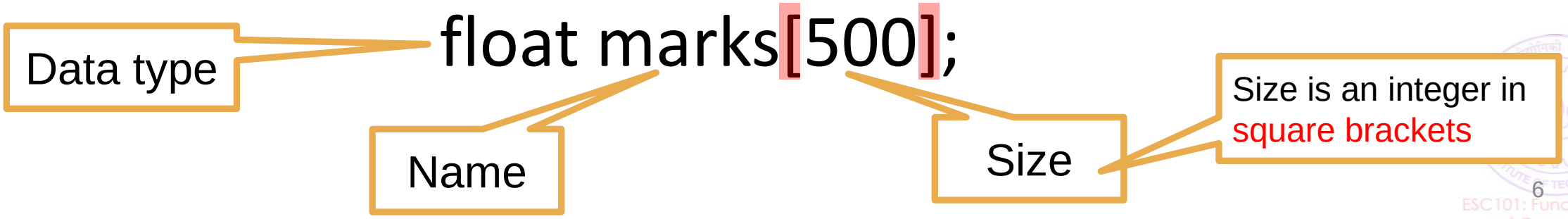
- Consider storing Major Quiz 1 marks of all ESC101 students

```
float marks1, marks2, marks3, .... ,marks500;
scanf("%f", &marks1);
scanf("%f", &marks2);
...
scanf("%f", &marks500);
```

- This is very time consuming, inelegant, and boring
- Arrays provide a better and more efficient way of doing such things

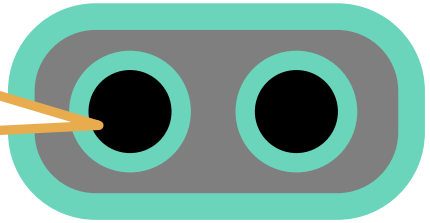
Arrays

- Not a new data type but a "data structure" (a collection of variables)
- Enables storing multiple values of the same data type
- Specification/declaration of an array needs the following
  - A data type (data type of values to be stored; all must be of same type)
  - A name (same naming convention as variables)
  - A size (how many values to be stored)
- An example of array declaration:



Arrays: Basic Syntax

In some other languages (such as Python), array indexing starts with 1



- Each value within the array is called "element" of the array
- Each element is accessed using a non-negative integer-valued **index**
  - First index is 0 (index of first element)
  - Last index is array size minus one (index of last element)
- Syntax to access/use each element of the array

  name_of_array[index]

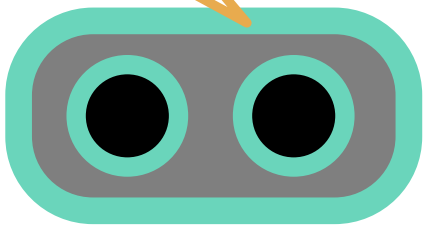
  Index is an integer in square brackets
- For example:
  - marks[0] is the first element of an array named marks
  - marks[1] is the second element of array marks
  - marks[499] is the last element of array marks

Arrays: Basic Syntax

- Array index needs to be within limits (0 to array size - 1)
- For an array declared as marks[500]
  - Index -1 is invalid (may give segmentation fault, also known as "segfault")
  - Index 510 is invalid (may give segfault)
  - Index 500 is also invalid (recall that 499 is the index of last element)
- Array index need not be a constant integer,
  - Can also be a variable/expression (but only int)
  - Example: marks[i] or marks[2*i + 1] where i is an integer
- Never use a float/double as index

Arrays: Storage in memory

Array name is like street name, index is like house number



- A single variable (int/float/char etc) is stored in a single box in memory

```
int a  = 2;
float b = 3.2;
```

- An array is stored in several consecutive boxes in memory (number of these boxes is equal to the size of the array)

```
float marks[500];
```

- More on storage of arrays when we study Pointers

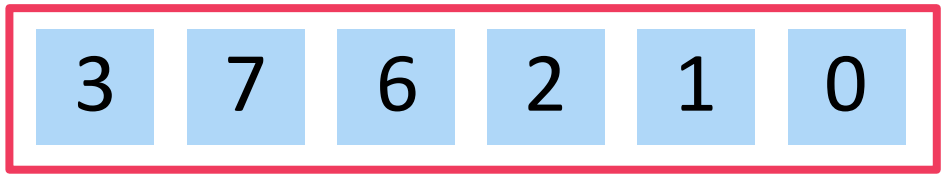
Array: Declaration and Initialization

Array elements are comma separated

Can be initialized at time of declaration itself

```
int a[6] = {3,7,6,2,1,0};
```

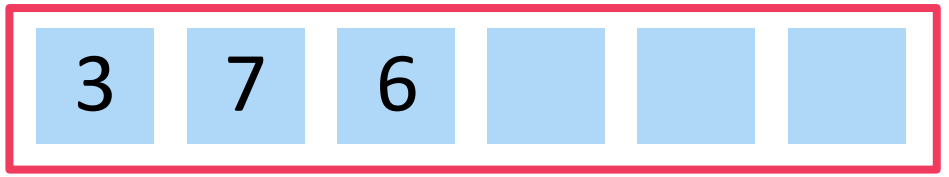
a



Can be partly initialized as well

```
int a[6] = {3,7,6};
```

a



However, if not initialized in same line with declaration, have to be initialized one by one!

```
int a[6];
a = {3,7,6,2,1,0};
```

```
int a[6];
a[2] = 6;
```

Array: Declaration and Initialization

Can be initialized at time of declaration itself

```
int a[6] = {3,7,6,2,1,0};
```

a

3	7	6	2	1	0

Can be partly initialized as well

```
int a[6] = {3,7,6};
```

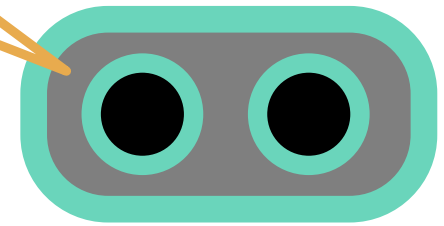
a

3	7	6			

Over initialization may crash

```
int a[6] = {1,2,3,4,5,6,7,8,9};
```

I will figure out how much space needed



No need to specify the array size during declaration

Better way is the following

```
int a[] = {1,2,3,4,5,6,7,8,9};
```

Warning: uninitialized arrays contain garbage, not zeros

Array: Traversal

```
#include <stdio.h>

int main()
{
        int array[] = {1,2,3,4,5};
        int i, n = 5;
        printf(" The array elements are: \n " );
        for( i=0;i < n; i++)
        {
                printf(" array[%d] = %d \n " , i, array[i] );
        }
        return 0;
}
```

Initialize

Access

Output:

```
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
```

* Pro-tip: Remember that first element has index 0, and nth element has index n-1

Next Class: More on arrays..

- Other ways of initializing arrays
- Example programs using arrays
- Array functions