Programming with C I

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2024.01.xx

Introduction to Arrays

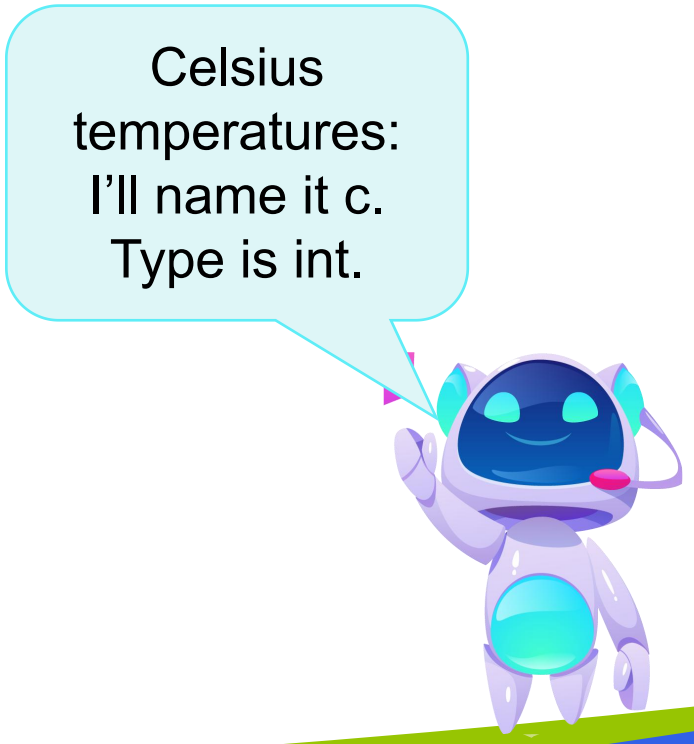
A collection of variable data

- Same name
- Same type
- Contiguous block of memory

Can manipulate or use

- Individual variables or
- ‘List’ as one entity

-45
6
0
72
1543
-89
0
62
-3
1
66453
78



Introduction to Arrays

- Used for lists of like items
  - Scores, speeds, weights, etc.
  - Same type
  - Avoids declaring multiple simple variables
- Used when we need to keep lots of values in memory
  - Sorting
  - Determining the number of scores above/below the mean
  - Printing values in the reverse order of reading
  - Etc.

Declaring Arrays

- General Format for declaring arrays

```
<data type>  <variable>  [<size>];
```

- Declaration
  - Declaring the array→allocates memory
  - Static entity - same size throughout program
- Examples:

```
int c[12];
int scores[300]
float weight[3284];
char alphabet[26]
```

Type is int.
Name is c.

Defined Constant as Array Size

- Use defined/named constant for array size
  - Improves readability
  - Improves readability
  - Improves maintainabilit
- Examples:

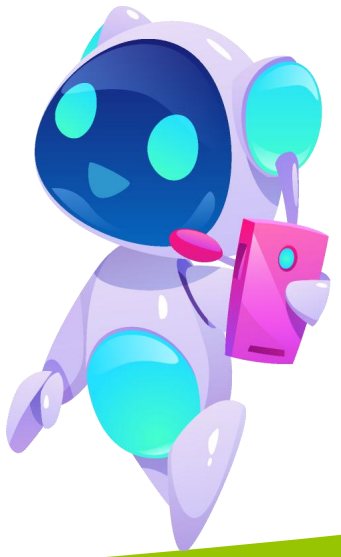
```
const int NUMBER_OF_STUDENTS = 50;
// ..
int scores[NUMBER_OF_STUDENTS];
```

```
#define NUMBER_OF_STUDENTS 50;
// ..
int scores[NUMBER_OF_STUDENTS];
```

Powerful Storage Mechanism

- Can perform subtasks like:
  - "Do this to i-th indexed variable"

    where i is computed by program
  - "Fill elements of array scores from user input"
  - "Display all elements of array scores“
  - “Sort array scores in order”
  - “Determine the sum or average score”
  - "Find highest value in array scores"
  - "Find lowest value in array scores"



Accessing Array Elements

- Individual parts called many things:
  - Elements of the array
  - Indexed or subscripted variables
- To refer to an element:
  - Array name and subscript or index
  - Format: **arrayname[subscript]**
- Zero based
  - **c[0]** refers to c_0, c sub zero, the **first** element of array c

Name of array (note that all elements of this array have the same name, c)

c [0]	-45
c [1]	6
c [2]	0
c [3]	72
c [4]	1543
c [5]	-89
c [6]	0
c [7]	62
c [8]	-3
c [9]	1
c[10]	66453
c[11]	78

Position number of the element within array c

Accessing Array Elements

- Example

```
Printf("%d\n", c[5];
```

- Note two uses of brackets:
  - In declaration, specifies SIZE of array
  - Anywhere else, specifies a subscript/index

Accessing Array Elements

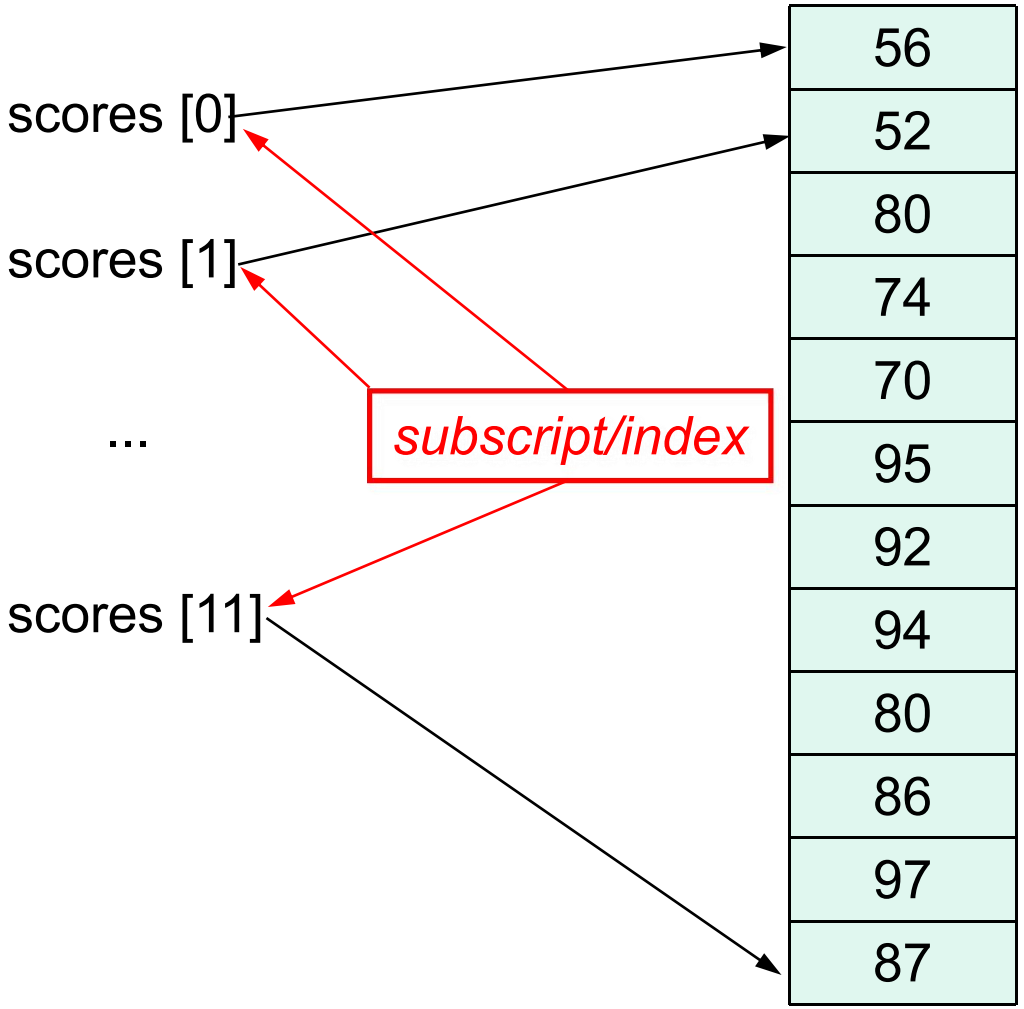
Example

- Given the declaration

```
int scores[12];
```

  - We reference elements of scores by

```
// Given these element values
// What does this print?
printf("%d\n", scores[3]);
```



Accessing Array Elements

- Size, subscript need not be literal constant
  - Can be named constant or expression

```
int scores[MAX_SCORES];  // MAX_SCORES is a constant
scores[n+1] = 99;        // If n is 2, same as scores[3]
```

Major Array Pitfall

- Array indexes go from 0 through size-1!
- C will 'let' you go out of the array's bounds
  - Unpredictable results – may get segmentation fault
  - Compiler will not detect these errors!
- Up to programmer to 'stay in bounds'

```
printf("%d\n", scores[-8]);
scores[250] = 88;
```

56
52
80
74
70
95
92
94
80
86
97
87

Initializing Arrays

- Arrays can be initialized at declaration

```
int scores[3] = {76, 98, 83};
```

  - Size cannot be variable or named constant
  - Equivalent to

```
int scores[3];
scores[0] = 76;
scores[1] = 98;
scores[2] = 83;
```

Auto-Initializing Arrays

- If fewer values than size supplied:
  - Fills from beginning
  - Fills 'rest' with zero of array base type
    - Declaration

```
int scores[5] = {76, 98, 83};
```

    - Performs initialization

```
scores[0] = 76;
scores[1] = 98;
scores[2] = 83;
scores[3] = 0;
scores[4] = 0;
```

Auto-Initializing Arrays

- If array size is left out
  - Declares array with size required based on number of initialization values
  - Example:

```
int scores[ ] = {76, 98, 83};
```

    - Allocates array scores with size of 3

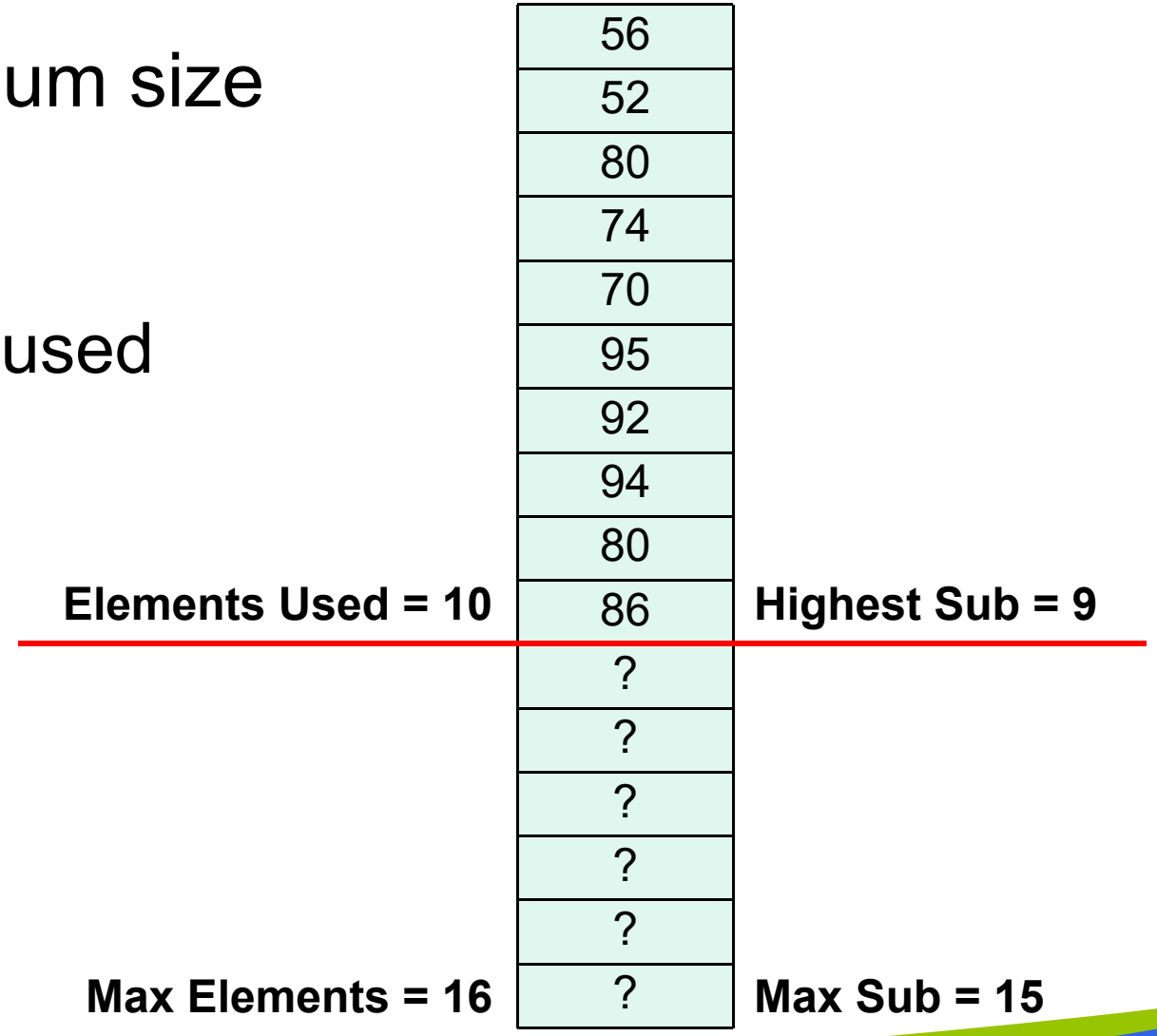
Partially Filled Arrays

- A program may need to process many lists of similar data but the lists may not all be the same length.
- In order to reuse an array for processing more than one data set, you can declare an array large enough to hold the largest data set anticipated.
- Then your program should keep track of how many array elements are actually in use.

Partially-filled Arrays (Common Case)

- Must be declared some maximum size
- Program must maintain
  - How many elements are being used

    and/or
  - Highest subscript



THE END

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2024.02.14