LECTURE 11 :
ARRAY
DR. ALSHIMAA HAMDY

Arrays

Content

- One-dimensional arrays
- Array elements
- Approaches of initialization arrays
- Out-of-Bound error
- Array of characters
- Passing arrays to functions
- Searching arrays
- Multidimensional arrays

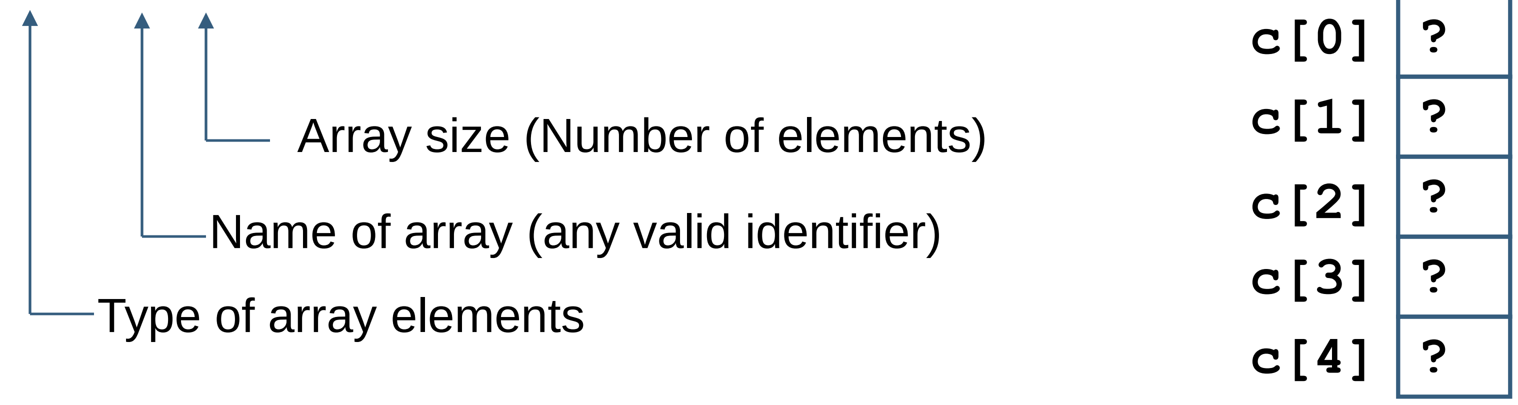
Array

- Data structure containing related data items of the same type
  - Hold in consecutive group of memory locations

- "static" entity remain the same size throughout program execution

- Syntax: `dataType arrayName[arraySize];`

  - `int c[5];`


Array size (Number of elements)
Name of array (any valid identifier)
Type of array elements
c[0] ?
c[1] ?
c[2] ?
c[3] ?
c[4] ?

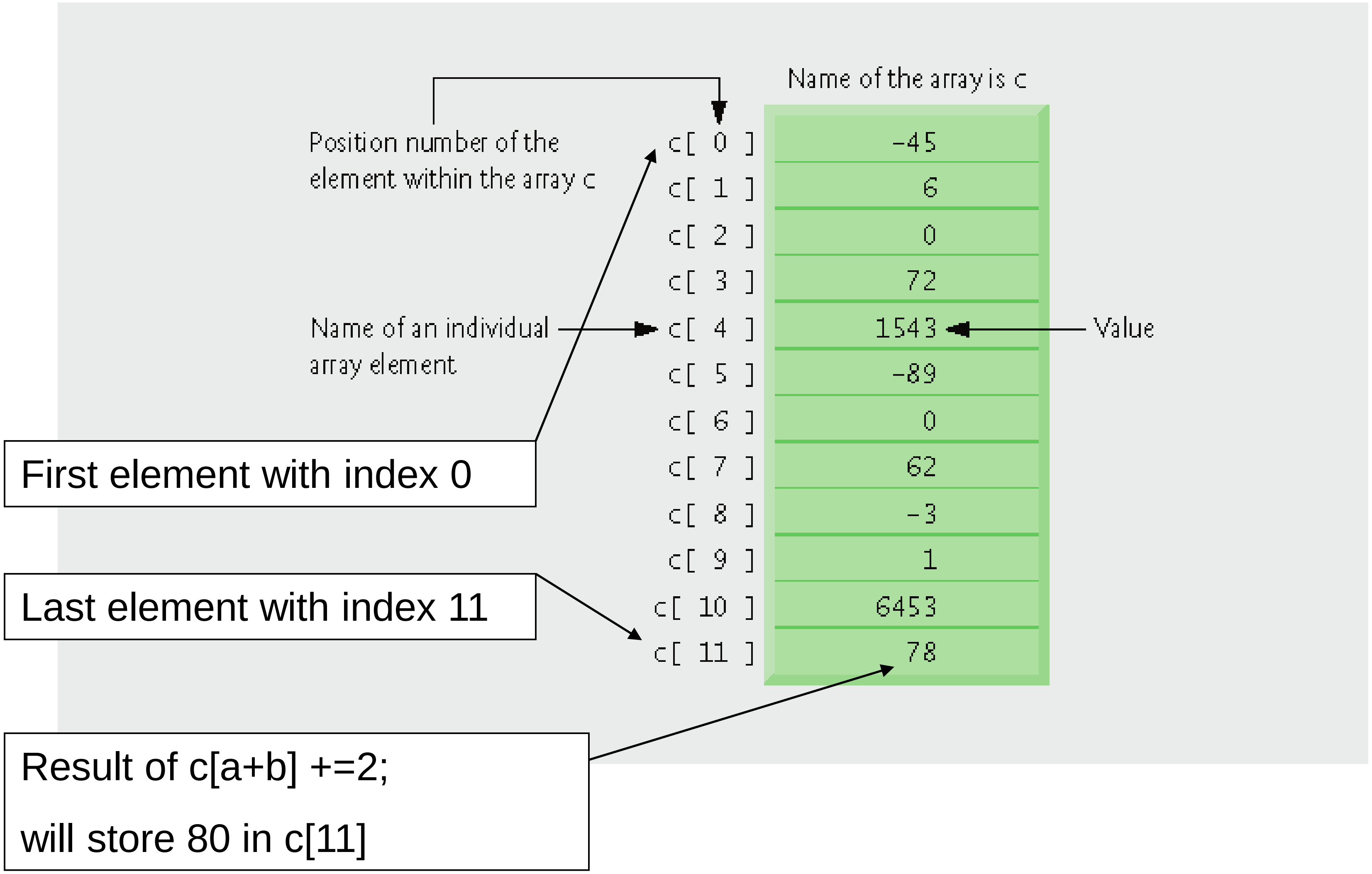
Array Elements

- Data items in the array
- Each element has a position in the array called index or subscript
- First element has index 0 and last element has index arraySize-1
- To access any element, giving array name followed by index in [ ]
  - `c[0], c[1],` …etc
  - [ ] is an operator has same precedence and associativity of ( )

- Position number must be
  - Positive integer **OR** any expression that results in a positive integer

```
int a=5, b=6;
c[a+b] +=2; // Adds 2 to array element c[ 11 ]
```

Array Elements (cont.)



First element with index 0

Last element with index 11

Result of c[a+b] +=2;

will store 80 in c[11]

Array Initialization –

- In declaration by using initializer list
  - Initializer list
    - Items enclosed in braces {}
    - Items in list separated by commas
    - Items themselves called initializers

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

c[0]	10
c[1]	20
c[2]	30
c[3]	40
c[4]	50

- If initializers are **<u>more</u>** than array elements (size)
  - Produce a compilation error

```
int c[5] = { 10, 20, 30, 40, 50, 60, 70 };
```

Array Initialization –

- If initializers are **fewer** than array elements (size)
  - Remaining elements are initialized to zero

    ```
    int c[5] = { 11 };
    ```

- To initialize all array elements to 0
  - Explicitly initializes first element to zero
  - Implicitly initializes remaining four elements to zero

    ```
    int x[5] = { 0 };
    ```

  - Static array is initialized to zero by compiler
    - No need to initialized explicitly

`c[0]`	11
`c[1]`	0
`c[2]`	0
`c[3]`	0
`c[4]`	0

`x[0]`	0
`x[1]`	0
`x[2]`	0
`x[3]`	0
`x[4]`	0

Array Initialization –

- Omitting array size in declaration, compiler determines array size based on number of items in initializer list

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

  - How many elements in array c ? **OR** Array size?
  - Index values?
  - Initialized values?

Declare Array

```
int myNumbers[] = {25, 50, 75, 100};
int i;
for (i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

Declare Array

```
// Declare an array of four integers:
int myNumbers[4];

// Add elements
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;
```

Sizeof()

```
int myNumbers[] = {10, 25, 50, 75, 100};
printf("%lu", sizeof(myNumbers)); // Prints 20
```

Sizeof()

```
int myNumbers[] = {10, 25, 50, 75, 100};
int length = sizeof(myNumbers)
/ sizeof(myNumbers[0]);

printf("%d", length);  // Prints 5
```

Sizeof()

```
int myNumbers[] = {25, 50, 75, 100};
int length = sizeof(myNumbers) /
sizeof(myNumbers[0]);
int i;

for (i = 0; i < length; i++) {
  printf("%d\n", myNumbers[i]);
}
```

Array String–

```
int main() {
  char greetings[] = "Hello World!";
  printf("%s", greetings);

  return 0;
}
```

Array String–

```
#include <stdio.h>
int main() {
  char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l',
'd', '!', '\0'};
  char greetings2[] = "Hello World!";
    printf("%s\n", greetings);
  printf("%s\n", greetings2);
    return 0;
}
```

Array String–

```
#include <stdio.h>
int main() {
  char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l',
'd', '!', '\0'};
  char greetings2[] = "Hello World!";
    printf("%s\n", greetings);
  printf("%s\n", greetings2);
    return 0;
}
```

Array String–

```
#include <stdio.h>
int main() {
  char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l',
'd', '!', '\0'};
  char greetings2[] = "Hello World!";
    printf("%s\n", greetings);
  printf("%s\n", greetings2);
    return 0;
}
```

Array String–

```
int main() {
  char carName[] = "Volvo";
  int i;
    for (i = 0; i < 5; ++i) {
    printf("%c\n", carName[i]);
  }
  return 0;
}
```

Initialization of Multidimensional Array

- Can be initialized in declaration much like one-dimension array
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
    - 1 and 2 initialize `b[ 0 ][ 0 ]` and `b[ 0 ][ 1 ]`
    - 3 and 4 initialize `b[ 1 ][ 0 ]` and `b[ 1 ][ 1 ]`
  - `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`
    - Row 0 contains values 1 and 0 (implicitly initialized to zero)
    - Row 1 contains values 3 and 4
  - `int b[ 2 ][ 2 ] = { 1, 2, 3, 4 };`
    - 1 and 2 initialize `b[ 0 ][ 0 ]` and `b[ 0 ][ 1 ]`
    - 3 and 4 initialize `b[ 1 ][ 0 ]` and `b[ 1 ][ 1 ]`

thanks