# COSC 2436: Arrays

J. Eoin Donovan - 2023

# What is an array?

**An array is a series of elements of the same type stored together in a sequence.**

| | | | |
|---|---|---|---|
| ‘h’ | ‘r’ | ‘k’ | ‘e’ |

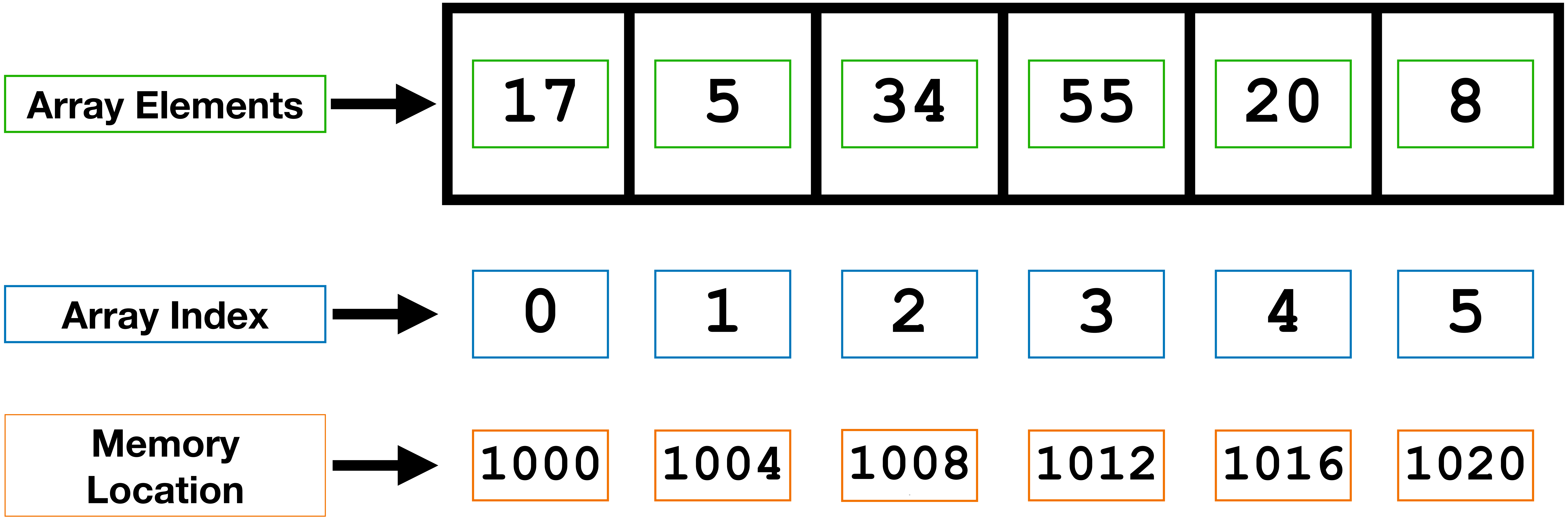
an array of characters

| | | | |
|---|---|---|---|
| 🍎 | 🍎 | 🍎 | 🍎 |

an array of apples

# How do arrays work?

Arrays work by allocating a contiguous block of memory to store elements in sequential order.



| | | | | | | |
|---|---|---|---|---|---|---|
| Array Elements | 17 | 5 | 34 | 55 | 20 | 8 |
| Array Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Memory Location | 1000 | 1004 | 1008 | 1012 | 1016 | 1020 |

# When should you use an array?

**You should use an array when you want to store multiple values of the same type and you know how many values you need stored.**

Avg. Temperature for the Week

| | | | | | | |
|---|---|---|---|---|---|---|
| 72° | 75° | 79° | 78° | 81° | 78° | 76° |

# What are advantages of arrays?

- **Random Access to Elements - you can access any element directly using its index**

```
                    array[3] = 213;
```

- **Easy Iteration - index is based on position in array**

```
        for(int i = 0; i < array_size; i++){
            ...
        }
```

- **Create Multi-Dimensional Arrays - easy to create arrays of differnet dimmensions**

```
                    int matrix[1][2];
```

# What are disadvantages of arrays?

- **Fixed Size - you must know the size when initializing array**

```
int array[5];
~~array[6] = 17;~~
```

- **Memory Wastage - number of elements might be less than array size**

```
int gradesArray[10000];
```

- **Shifting - if you insert/delete in the middle, you have to shift other elements in the array**

```
{1, 2, 3} => delete(2) => {1,  , 3} => {1, 3,  }
```

# What are the types of arrays?

There are two types of arrays static and dynamic.

Static Arrays are allocated memory at compile-time and their size is fixed and determined at the point of declaration.

Dynamic Arrays are allocated memory at runtime. The size of a dynamic array can be determined and changed in different parts of the code.

# How do you create a static array?

**To create a static array you first declare the type, then give the array a name, and finally declare the size of the array inside brackets.**

```
int arr[10];
```

# How do you create a dynamic array?

**To create a dynamic array follow the style below:**

```
type *name = new type[size];
```

**Examples:**

```
int *arr = new int[10];
char *arr = new char[24];
```

# How do you create a dynamic array?

**With a dynamic array you also need to remember to delete it too. You can accomplish that by doing:**

```
delete [] name;
```

**Example:**

```
int *arr = new int[10]; //Creates array

delete [] arr; //Deletes array
```

# Array Practice: Find Max

**Write the function `int findMax(int arr[], int size)` which returns the maximum value in an array. Assume `size` >= 1.**

**Input:** `arr = {3, 7, 12, 10, 9}`, `size = 5`
**Output:** `12`

```
int findMax(int arr[], int size){




}
```

# Array Practice: Find Max (Solution)

```
int findMax(int arr[], int size){
  int max = arr[0];
  for(int i = 0; i < size; i++){
    if(arr[i] > max){
      max = arr[i];
    }
  }
  return max;
}
```

# Array Practice: Find Max (Solution)

```
int findMax(int arr[], int size){
  int max = arr[0];
  for(int i = 0; i < size; i++){
    if(arr[i] > max){
      max = arr[i];
    }
  }
  return max;
}
```

**Since we know `size` >= 1, we can set the `max` to `arr[0]` and then compare the rest of the array to this value to find the `max`.**

# Array Practice: Contains Duplicates

**Write the function `bool containsDuplicates(int arr[], int size)` which returns true if the array contains duplicates and false otherwise.**

**Input:** `arr = {4, 16, 7, 16, 20},size = 5`
**Output:** `true`

```
bool containsDuplicates(int arr[], int size){




}
```

# Array Practice: Contains Duplicates (Solution)

```
bool containsDuplicates(int arr[], int size){
  for(int i = 0; i < size; i++){
    for(int j = i + 1; j < size; j++){
      if(arr[i] == arr[j]){
        return true;
      }
    }
  }
  return false;
}
```

# Array Practice: Contains Duplicates (Solution)

```
bool containsDuplicates(int arr[], int size){
  for(int i = 0; i < size; i++){
    for(int j = i + 1; j < size; j++){
      if(arr[i] == arr[j]){
        return true;
      }
    }
  }
  return false;
}
```

**We set `j` to `i+1` to ensure we don’t look at the same element twice.**

# Array Practice: Contains Duplicates (Solution)

```
bool containsDuplicates(int arr[], int size){
  for(int i = 0; i < size; i++){
    for(int j = i + 1; j < size; j++){
      if(arr[i] == arr[j]){
        return true;
      }
    }
  }
  return false;
}
```

**Once both for loops end, we can return false because we will have checked the entire array for duplicate values.**

# Array Practice: Two Sum

**Write the function `void twoSum(int arr[], int size, int target)` which prints the indices of the two numbers that add up to `target`. Assume each input has exactly one solution and you cannot use the same element twice.**

**Input:** `arr = {2, 5, 8, 3}`, `size = 4`, `target = 10`
**Output:** `[0, 2]`

```
void twoSum(int arr[], int size, int target){




}
```

# Array Practice: Two Sum (Solution)

```
void twoSum(int arr[], int size, int target){
  for(int i = 0; i < size; i++){
    for(int j = i + 1; j < size; j++){
      if(arr[i] + arr[j] == target){
        cout << “[“ << i << “, “ << j << “]”;
        return;
      }
    }
  }
}
```

# Array Practice: Two Sum (Solution)

```
void twoSum(int arr[], int size, int target){
  for(int i = 0; i < size; i++){
    for(int j = i + 1; j < size; j++){
      if(arr[i] + arr[j] == target){
        cout << “[“ << i << “, “ << j << “]”;
        return;
      }
    }
  }
}
```

**Similar to the containsDuplicates problem, we set `j` to `i+1` to ensure we don’t check the same element twice.**

# Array Practice: Two Sum (Solution)

```
void twoSum(int arr[], int size, int target){
  for(int i = 0; i < size; i++){
    for(int j = i + 1; j < size; j++){
      if(arr[i] + arr[j] == target){
        cout << “[“ << i << “, “ << j << “]”;
        return;
      }
    }
  }
}
```

**Since we know there is only one solution, we can go ahead and return once we’ve found the solution.**

# Array Practice: Create 2D Array

**Write the code to create a dynamic, 2-dimensional array. The array should be of dimensions `10x10` (10 rows, 10 columns).**

# Array Practice: Create 2D Array Solution

```
int **arr = new int*[10];
for(int i = 0; i < 10; i++){
  arr[i] = new int[10];
}
```

# Array Practice: Delete 2D Array

**With the array you made in the previous problem, now write the code to delete the array.**

# Array Practice: Delete 2D Array Solution

```
for(int i = 0; i < 10; i++){
  delete [] arr[i];
}
delete [] arr;
```