Arrays

Arrays

- Used to store a collection of data

1	45	7	1000	-105	42

Arrays

- In C, the data are the same type
    - Ex: array can be all *ints*, or all *chars*, or all *doubles*, etc...

1	45	7	1000	-105	42

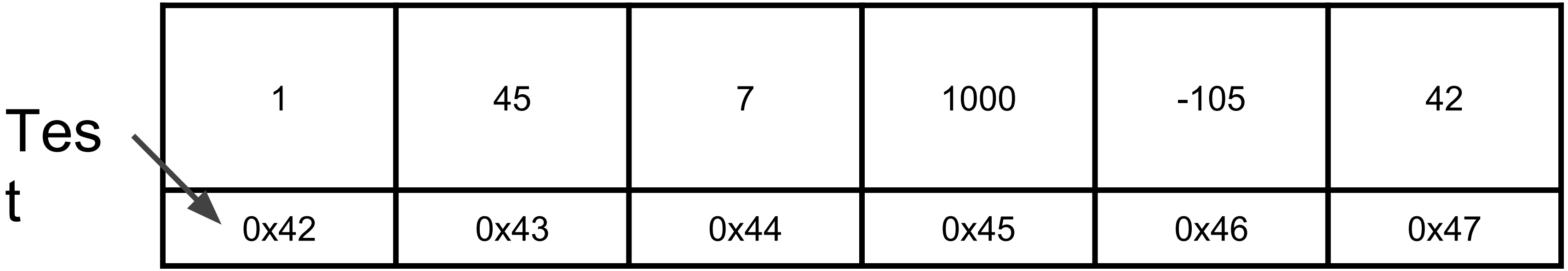
Arrays

- Each element within an array takes up an unique memory location
    - All memory locations are in sequence

1	45	7	1000	-105	42
0x42	0x43	0x44	0x45	0x46	0x47

Arrays

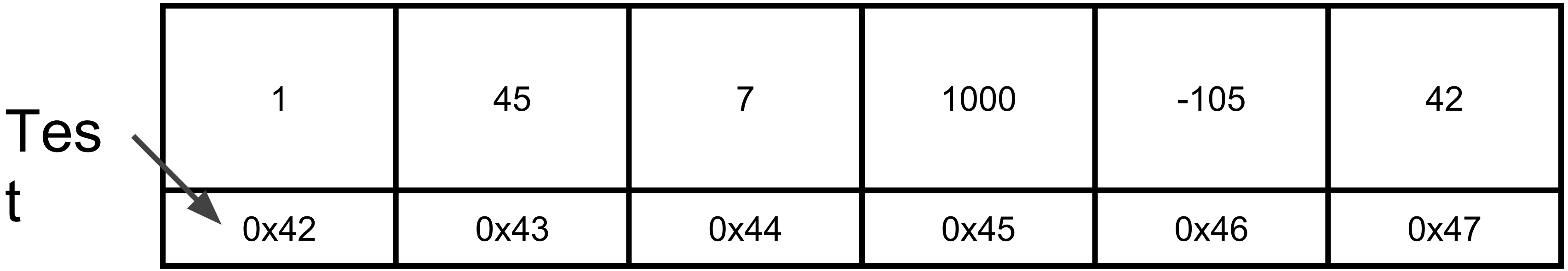
- The name of the array is a pointer to the first memory location
    - Ex: *test* points to 0x42



1	45	7	1000	-105	42
0x42	0x43	0x44	0x45	0x46	0x47

Arrays

- The name of the array is a pointer to the first memory location
  - Does not need & or * → points automatically as part of array definition



1	45	7	1000	-105	42
0x42	0x43	0x44	0x45	0x46	0x47

Arrays

- The values of the array are accessed through the *index* of an element

Tes
t

test[0]	test[1]	test[2]	test[3]	test[4]	test[5]
1	45	7	1000	-105	42
0x42	0x43	0x44	0x45	0x46	0x47

Arrays

- Array indexes start at 0, and go through length_of_array - 1
  - Ex: Test has 6 elements. Therefore, indexes go from 0 - 5

Tes
t

test[0]	test[1]	test[2]	test[3]	test[4]	test[5]
1	45	7	1000	-105	42
0x42	0x43	0x44	0x45	0x46	0x47

Declaring Arrays

- Static allocation
    - The memory used by the array is defined when the code is **compiled**
- Dynamic allocation
    - Memory used by the array (and other variables) is defined as the code is **running**

Declaring Arrays

- Static allocation
  - Used by C, C++, Java

- Dynamic allocation
  - Used by “newer” languages: Python, Matlab, etc…

Declaring Arrays

- Since C is statically allocated…
- ...the memory has be declared before using the array

```
> type name[size];
```

Declaring Arrays

```
> type name[size];
```

```
Ex:   > int test[10];

      > char name[20];

      > double gradez[50];
```

Arrays

- The name of the array is a pointer to the first memory location
  - Does not need & or * → points automatically as part of array definition

Tes
t

1	45	7	1000	-105	42
0x42	0x43	0x44	0x45	0x46	0x47

Declaring Arrays

- Can also populate the array at declaration
  - This kind of declaration uses *curly braces* instead of straight brackets

```
> int test[6] = {1, 1, 2, 3, 5, 8}
```

Declaring Arrays

- Can also populate individual elements of arrays by accessing the specific element

```
> int test[6];

> test[2] = 42;
```

Looping through Arrays

- For loops provide easy access to looping through arrays

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

Looping through Arrays

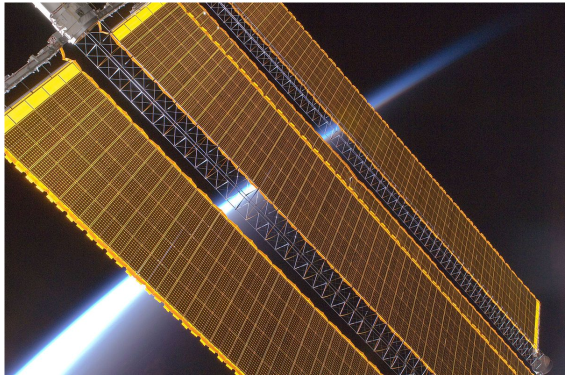
- For loops provide easy access to looping through arrays

```
for (i = 0; i < 10; i++) {
    printf("%d ", test[i]);
}
```

Accessing Array Variables

- Each array element is a unique variable
- Therefore, can use array elements as you would with normal variables

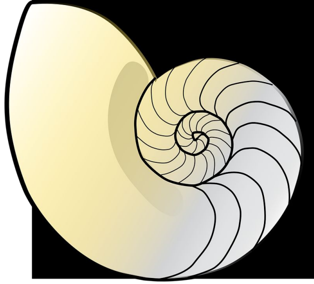
```
int num1 = array[0];

scanf(“%d”, &array[10]);

array[1] = array[0] * 2;
```



Array Coding Challenge 1

- Create an array, starting with the number 1, so that each following number is double the number before it
  - 1, 2, 4, 8, 16, etc…
  - Scan in the length of the array
  - Only the 0th element can be populated manually - the others must be calculated using the previous position in the array
    - Array[0] = 1

Array Coding Challenge 2



- Create an array that holds an arbitrary number of digits of the Fibonacci sequence
  - 1, 1, 2, 3, 5, etc…
- Restrictions
  - Can only define the first two elements manually (the others have to be calculated)
    - `Array[0] = 1; Array[1] = 1;`
  - Scan in the length, and use it to define the array
- Print out the elements of the array when completed