Engineering Technology (ENGR 101)

Arrays

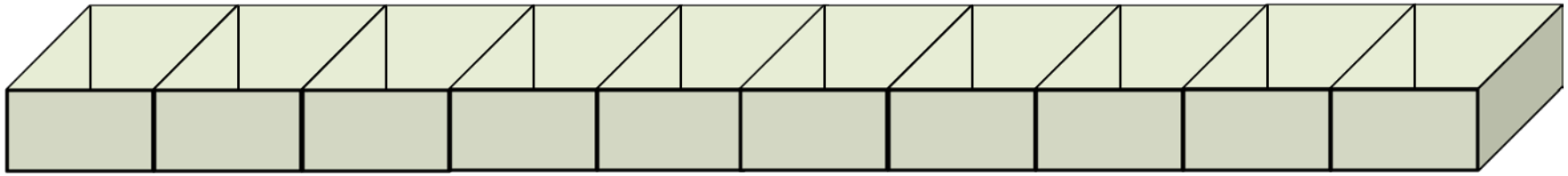
Mohammad Nekooei

Engineering and Computer Science

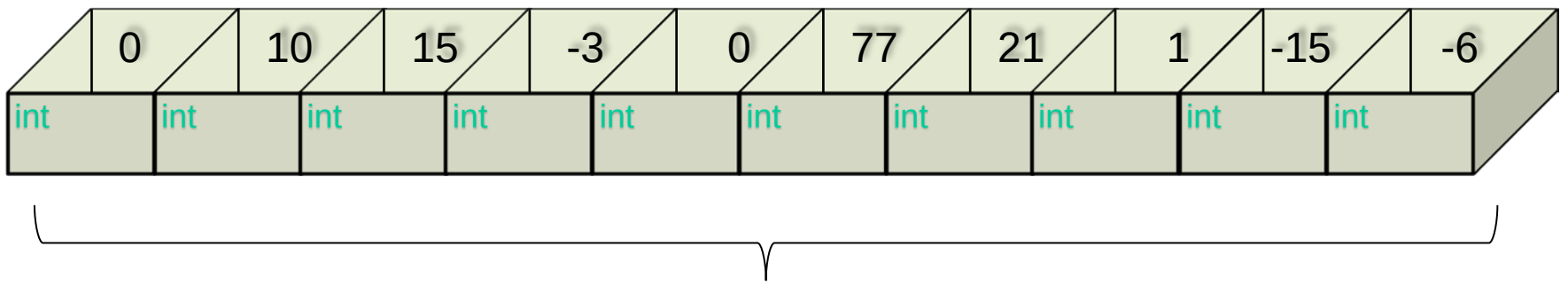
Victoria University of Wellington

Arrays

- An array is a collection of data that holds a **fixed** number of data (values) of the **same type**
  - Length determined when array is created
  - All elements are of the same type



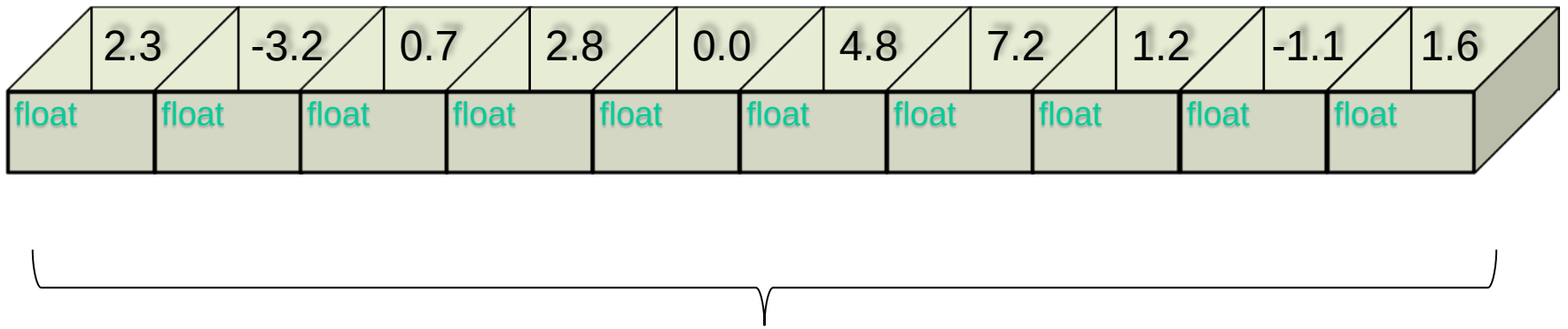
One-Dimensional Array Overview (1)



⇦ array of ints

size of array: 10

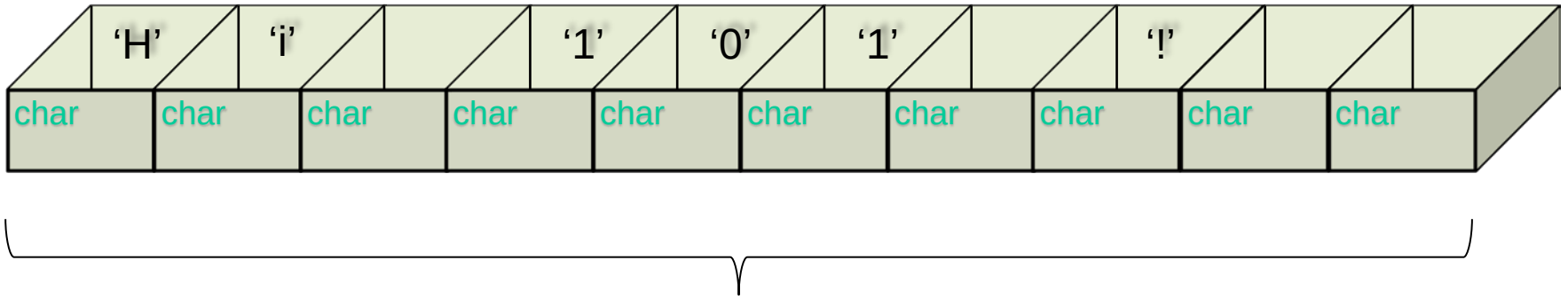
One-Dimensional Array Overview (2)



⇦ array of floats

size of array: 10

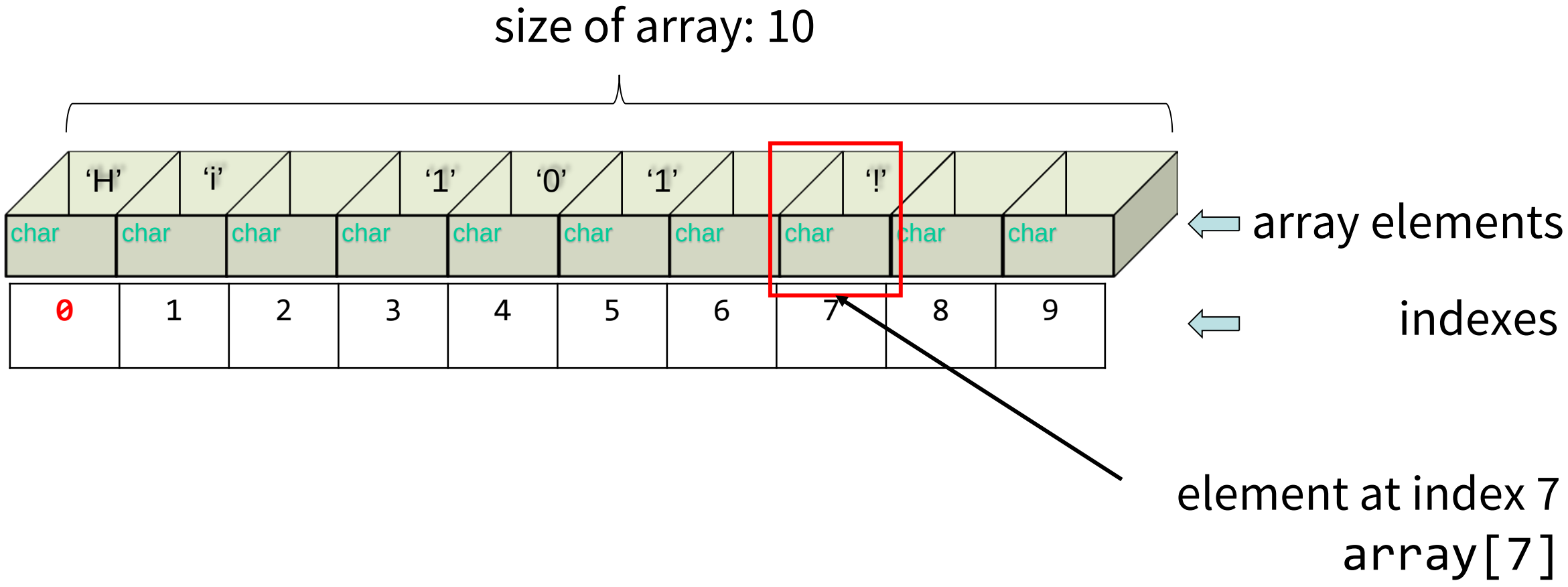
One-Dimensional Array Overview (3)



array of chars

size of array: 10

One-Dimensional Array Overview (4)



Arrays

- The simplest interpretation of an array is one-dimensional array, often referred to as a list

- The individual elements of the array can be accessed via **indices**
  - The first index of an array starts at `0`
  - If the size of an array is `n`, to access the last element the index `n-1` is used
  - This is because the index in C is actually an *offset* from the beginning of the array
    - The first element is at the beginning of the array, and hence has zero offset

Declaring Arrays

- Syntax for **declaring** an array:

```
data_type array_name[size];
```

- Example:
  - We declare an array named `data` of `float` type and size `4` as:

```
float data[4];
```

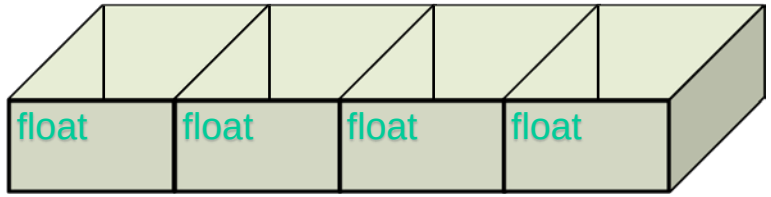
  - It can hold 4 floating-point values

- The **size** and **type** of arrays **cannot** be changed after their declaration!

Initializing Arrays (1)

- Arrays can be initialized **one-by-one**
- For example:

data

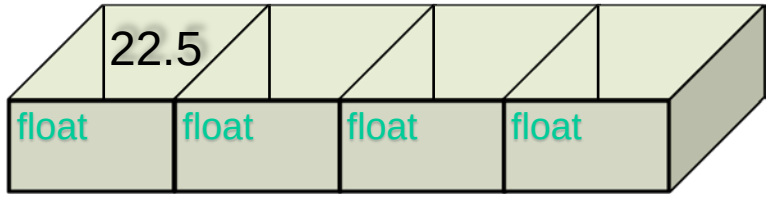


✓
```
float data[4];
data[0] = 22.5;
data[1] = 23.1;
data[2] = 23.7;
data[3] = 24.8;
```

Initializing Arrays (1)

- Arrays can be initialized **one-by-one**
- For example:

data

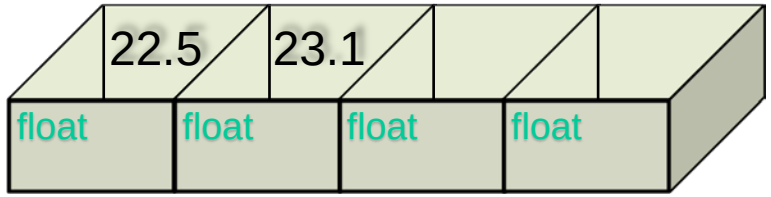


✓
```
float data[4];
```
✓
```
data[0] = 22.5;
data[1] = 23.1;
data[2] = 23.7;
data[3] = 24.8;
```

Initializing Arrays (1)

- Arrays can be initialized **one-by-one**
- For example:

data

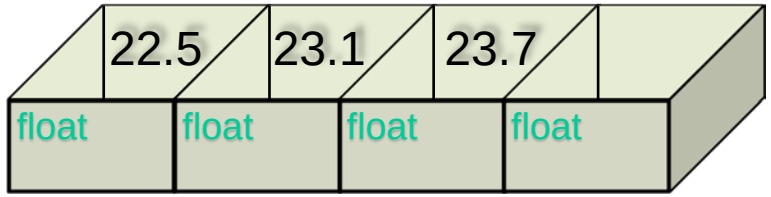



```
float data[4];
data[0] = 22.5;
data[1] = 23.1;
data[2] = 23.7;
data[3] = 24.8;
```

Initializing Arrays (1)

- Arrays can be initialized **one-by-one**
- For example:

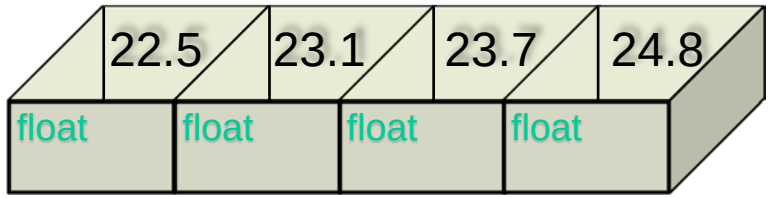
data




```
✓ float data[4];
✓ data[0] = 22.5;
✓ data[1] = 23.1;
✓ data[2] = 23.7;
  data[3] = 24.8;
```

Initializing Arrays (1)

- Arrays can be initialized **one-by-one**
- For example:



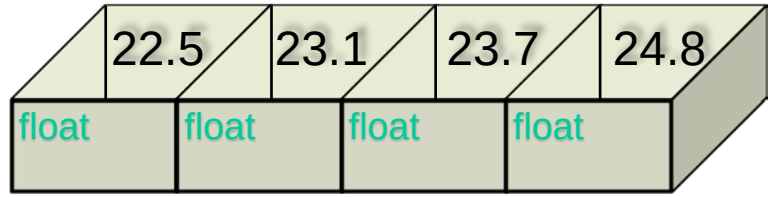

```
✓ float data[4];
✓ data[0] = 22.5;
✓ data[1] = 23.1;
✓ data[2] = 23.7;
✓ data[3] = 24.8;
```

- In the case of large arrays this method is inefficient

Initializing Arrays (2)

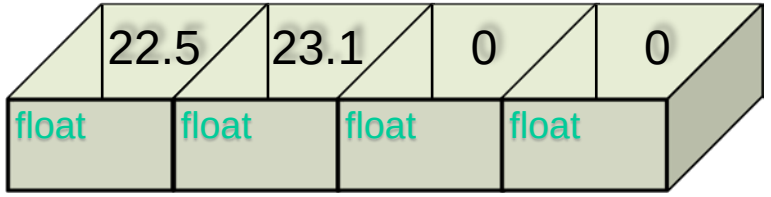
- Arrays can be also initialized when they are **declared** (just as any other variables):

```
float data[4] = {22.5, 23.1, 23.7, 24.8};
```



- An array may be **partially initialized**, by providing fewer data items than the size of the array

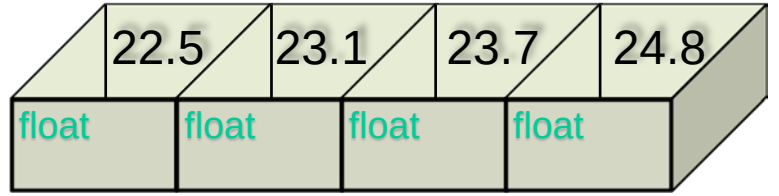
```
float data[4] = {22.5, 23.1};
```



  - The remaining array elements will be automatically initialized to zero

- If an array is to be completely initialized, the dimension (size) of the array is not required

```
float data[] = {22.5, 23.1, 23.7, 24.8};
```



  - The compiler will automatically size the array to fit the initialized data

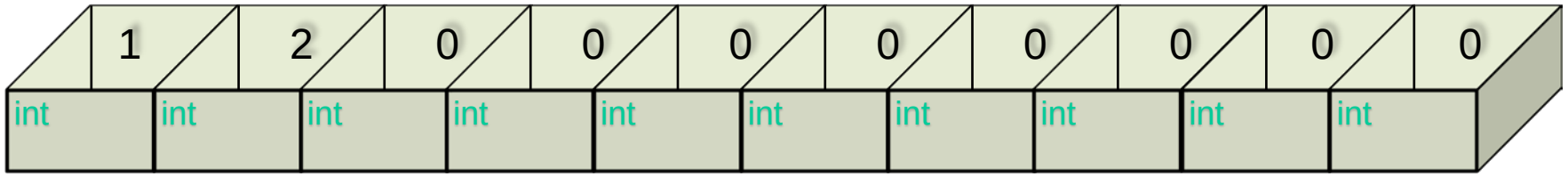
Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int array[10] = {1, 2};
```

```
int idx = 0;
while(idx < 10) {
      /* do something with array[idx] */
      idx++;
}
```

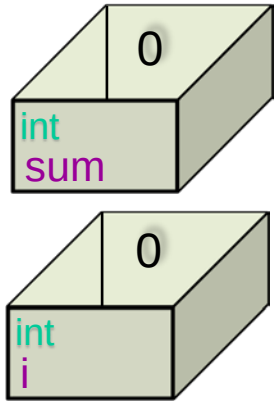
```
for (int idx = 0; idx < 10; idx++){
      /* do something with array[idx] */
}
```

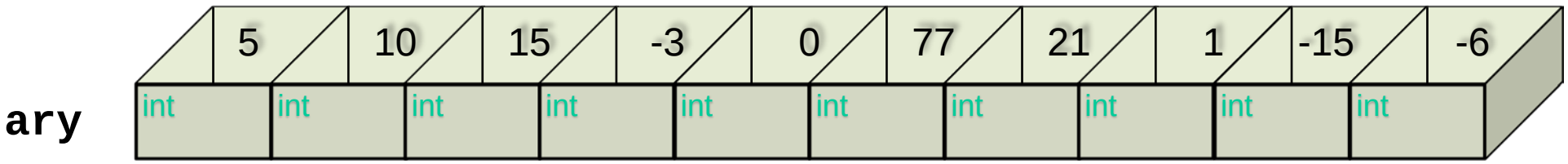


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

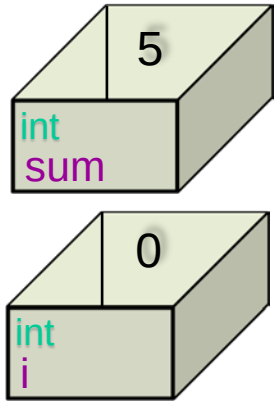


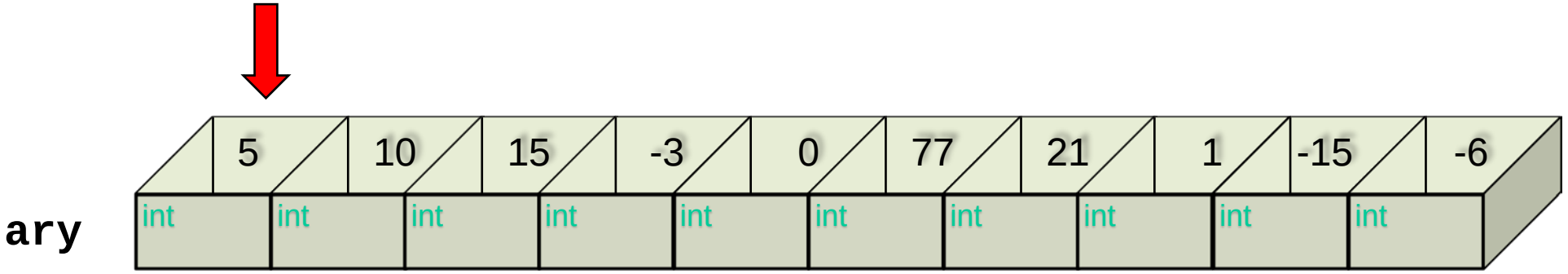


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

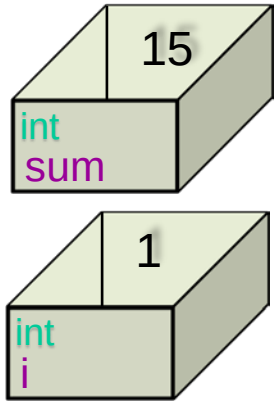


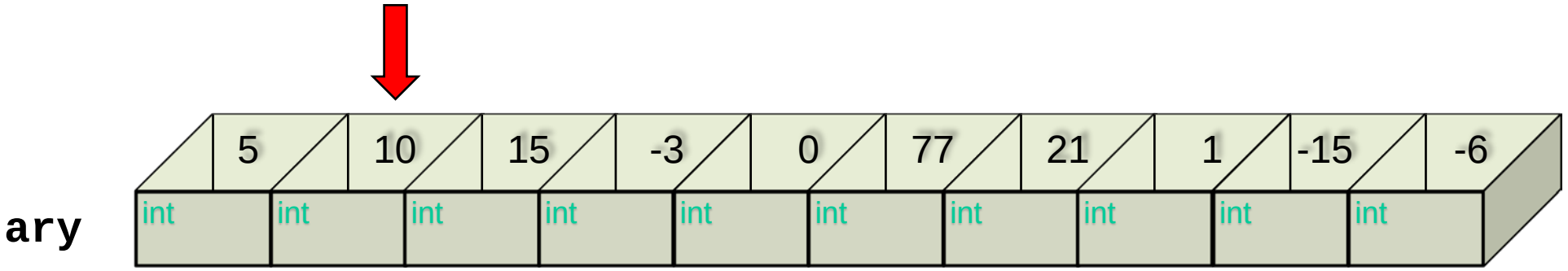


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

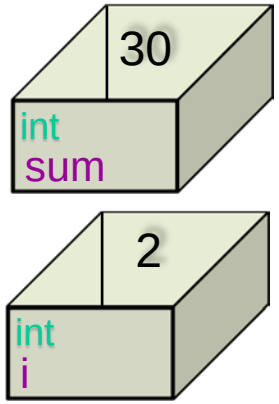


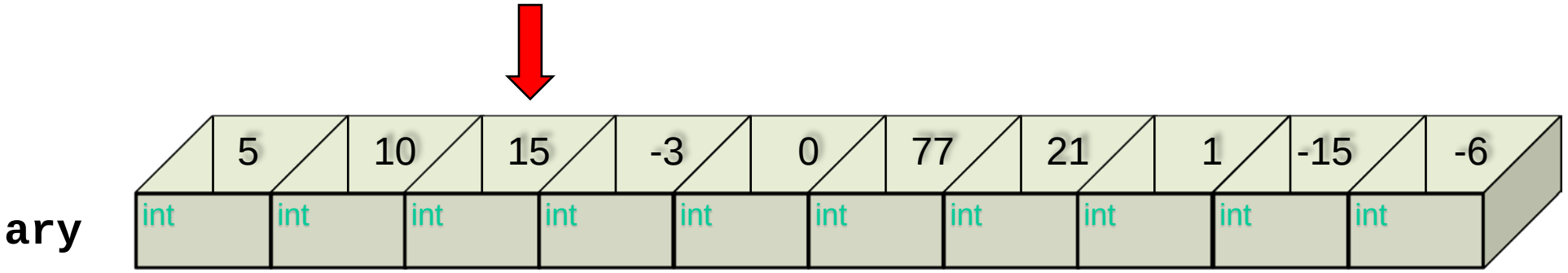


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

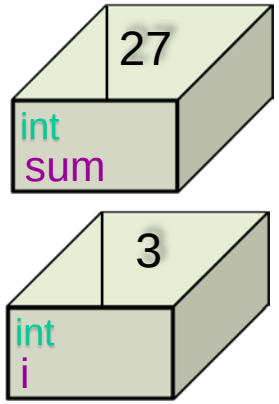


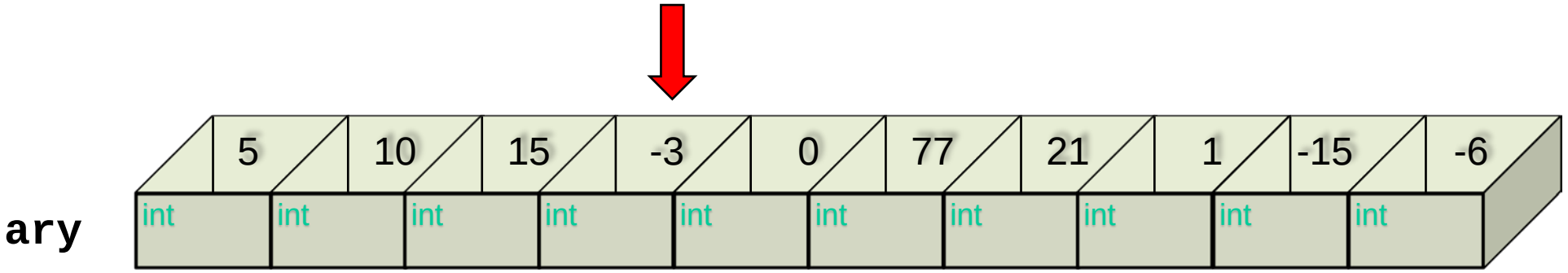


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

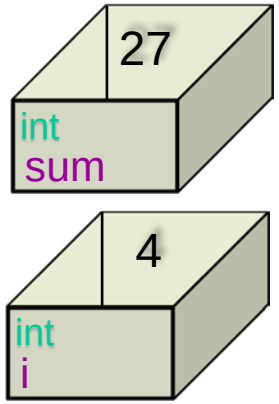


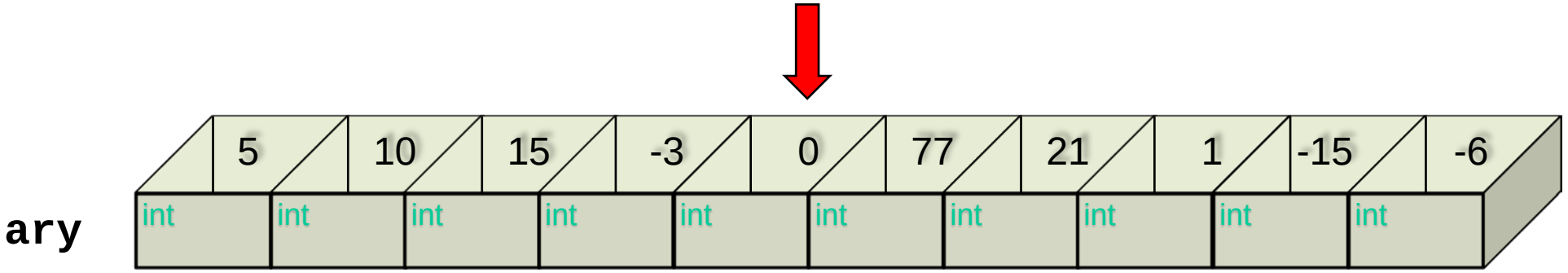


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

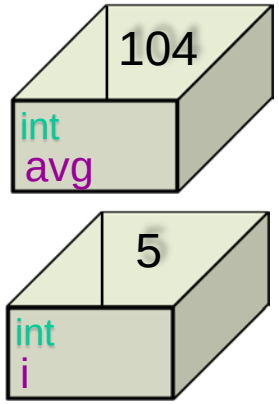


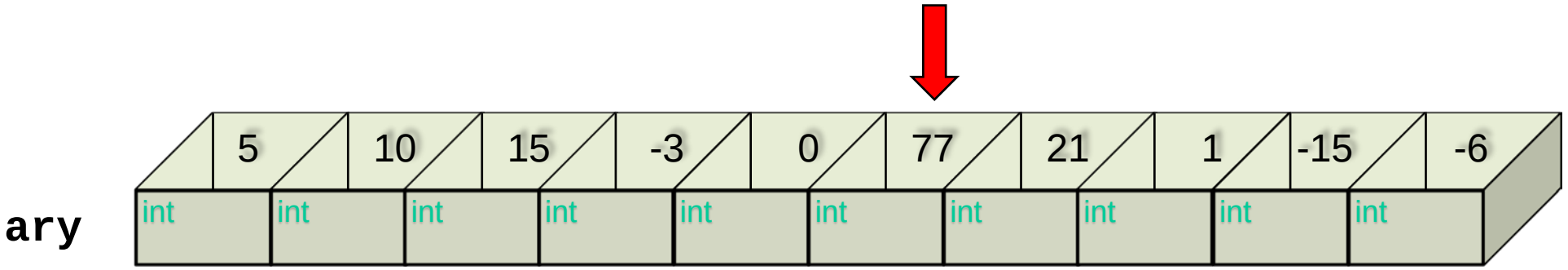


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

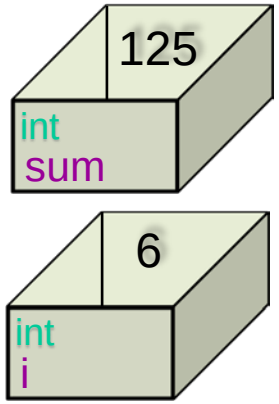


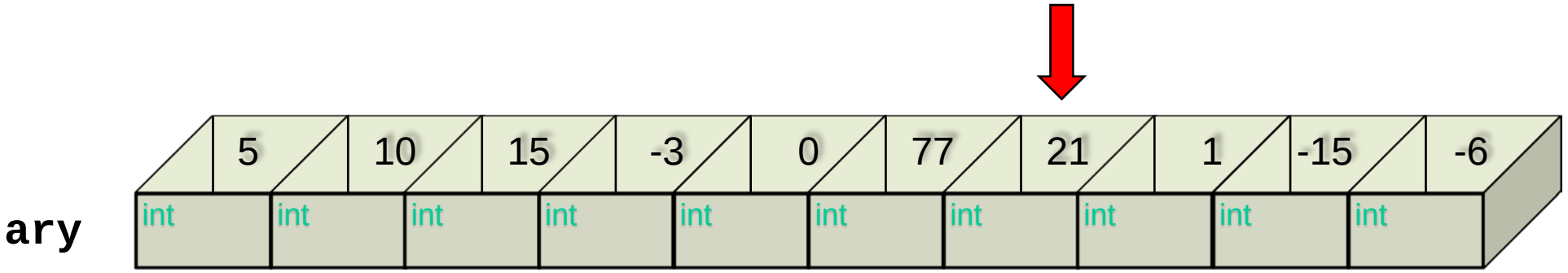


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

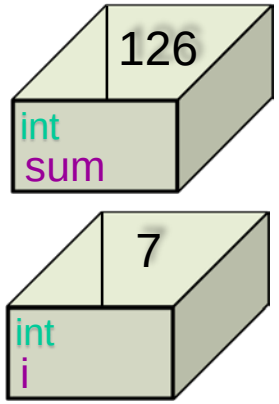


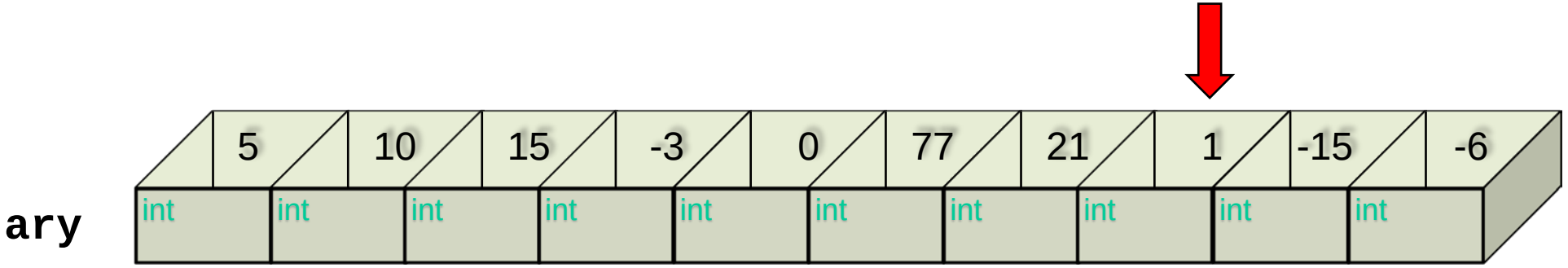


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

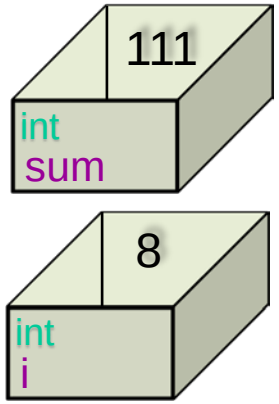


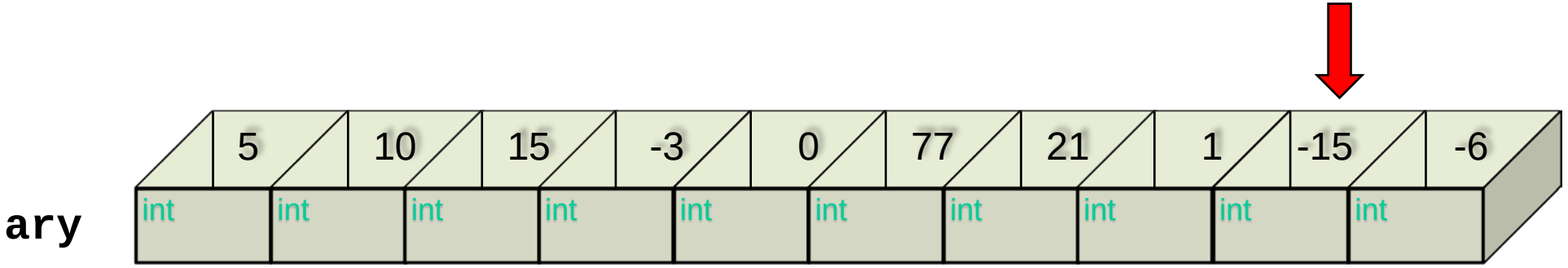


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```

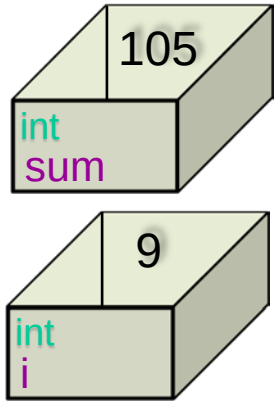


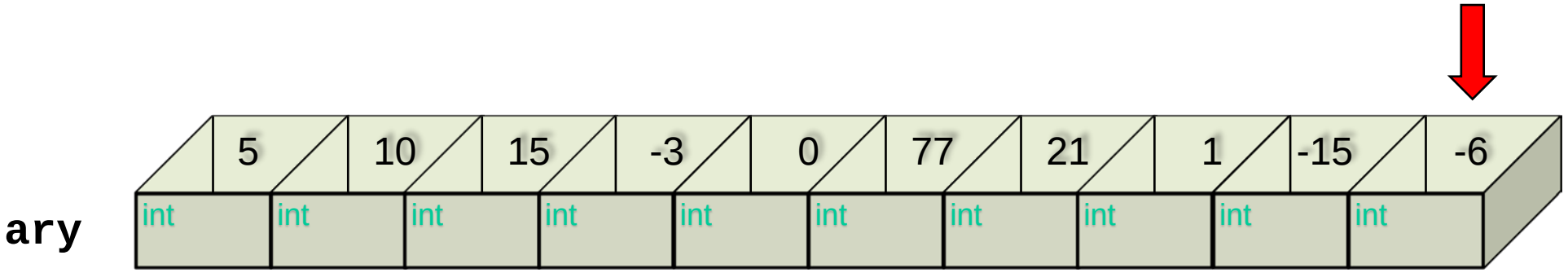


Arrays and Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:

```
int ary[10] = {5, 10, 15, -3, 0, 77, 21, 1, -15, -6};
int sum = 0;
for (int i = 0; i < 10; i++){
      sum = sum + ary[i];
}
Serial.println(sum);
```





Off-By-One Error

- The most common mistake when working with arrays in C is forgetting that indices start at `0` and stop one less than the array size
  - We often refer to this issue as "off-by-one error"

```
int data[]={1,2,3,4,5}; /* number of elements is 5 */

for (int idx = 0; idx <= 5; idx++){
      /* do something with data[idx] */
}
```

- The compiler does not control the limits of the array!