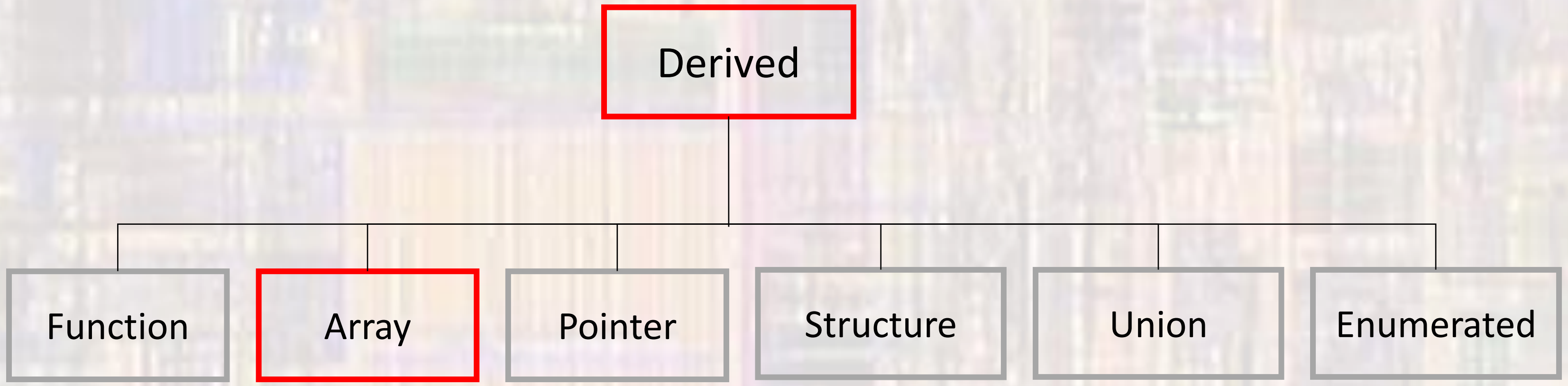
Arrays

Last updated 9/9/21

Arrays

- C Types
  - Arrays are a Derived type



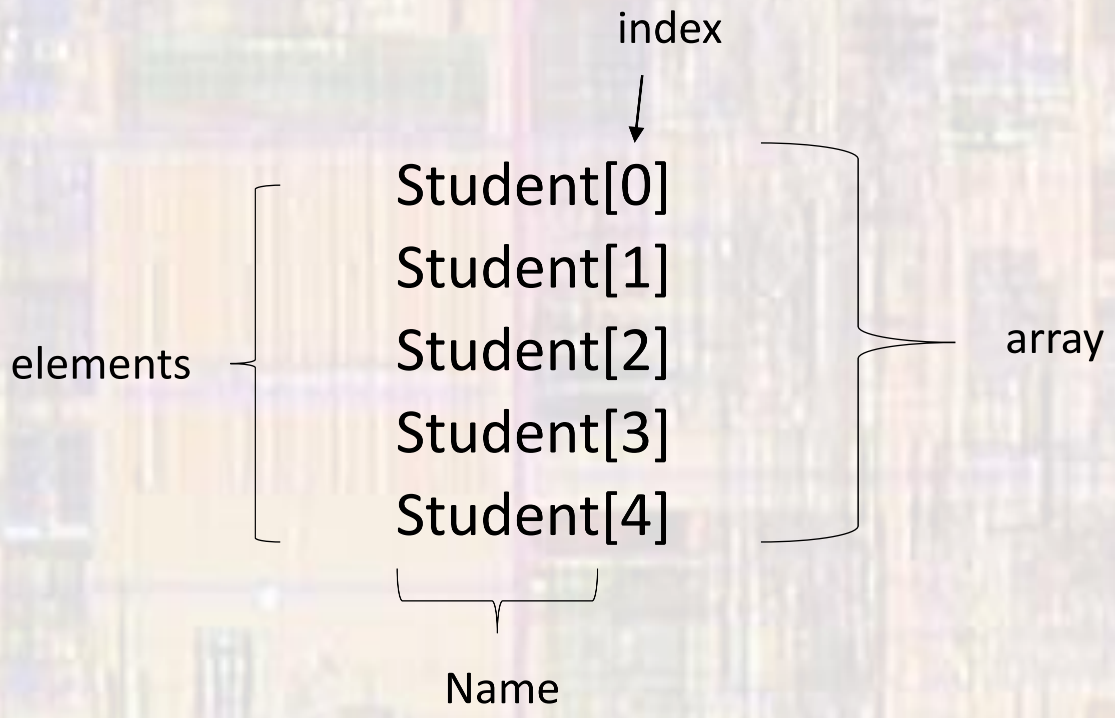
Arrays

- Arrays
  - Grouping of similar items

Student 0	Student_0	Student[0]
Student 1	Student_1	Student[1]
Student 2	Student_2	Student[2]
Student 3	Student_3	Student[3]
Student 4	Student_4	Student[4]

Arrays

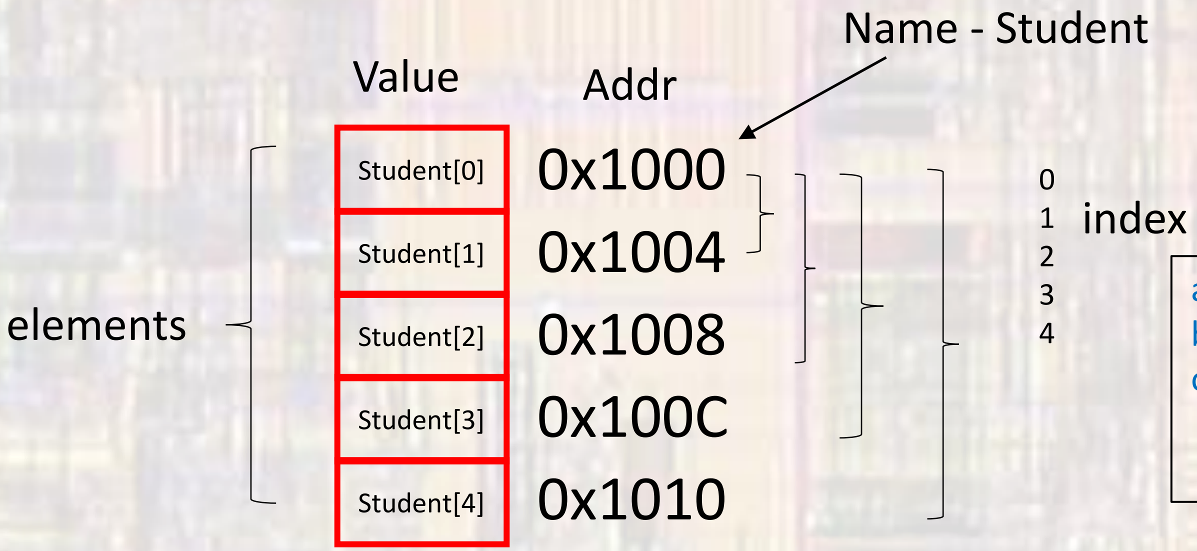
- Array notation



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

Arrays

- Array notation
  - In memory
  - Name is actually the address of the beginning of the array (a pointer)
  - Index is the offset from the name address
    - not an address



addr offset in memory is calculated by the compiler to match the size of the element types

offset = size_of_type * index

Arrays

- Arrays in C
  - All elements in the array must be of the same type

Why?

Arrays

- Arrays in C

  Declaration

  ```
  type arrayName[arraySize];
  ```

  Fixed length array – size known during compilation

  ```
  int scores[22];
  char first_name[15];
  ```

  Variable length array – size only known during execution

  ```
  float testAve[classSize];
  int numAs[gradesGE90];
  ```

  where classSize and gradesGE90 are integral variables

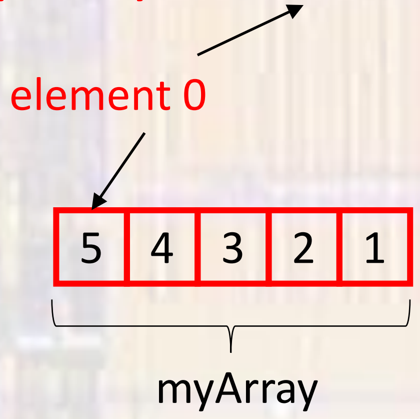
Arrays

- Arrays in C

  Declaration + Initialization

  ```
  type arrayName[arraySize] = {comma separated list};
  ```

  ```
  int myArray[5] = {5,4,3,2,1};        // basic
  ```



Arrays

- Arrays in C

  Declaration + Initialization

  type arrayName[arraySize] = {comma separated list};

```
int myArray[5] = {5,4,3,2,1};          // basic
```

5	4	3	2	1

```
int myArray[5] = {5,4};       // partial initialization
                              // others are set to 0
```

5	4	0	0	0

```
int myArray[ ] = {5,4,3,2,1};          // size is taken from
                                       // initialization values
```

5	4	3	2	1

```
int myArray[5] = {0};         // all set to 0
```

0	0	0	0	0

Arrays

- Arrays in C

  Variable length arrays **cannot** have an initialization

```
float testAve[classSize];
int numAs[gradesGE90];
```

Arrays

- Arrays in C

  Accessing elements

myArray	5	4	3	2	1

```
foo = myArray[3];          // foo = 2
foo = myArray[foo];        // foo = 3
```

```
myArray[0] = 0;
```

0	4	3	2	1

```
myArray[foo + 1] = 6;
```

0	4	3	2	6

Arrays

- Arrays in C
  - Keyboard example
    - Read the scores for 10 students from the keyboard and store them in an array

```
int scores[10];
int i;

for(i= 0; i < 10; i++){
    scanf(“%i”, &scores[ i ]);
}
```

Arrays

- Arrays in C
  - Keyboard example
    - Read the scores for 10 students from the keyboard and store them in an array

```
int scores[10];
int i;

for(i= 0; i < 10; i++){
    scanf("%i", &scores[ i ]);
}
```

note: still need '&' since
scores[i] is not a pointer (address)
It is an individual value

Arrays

- Arrays in C
  - Display example
    - Print the scores for 10 students from an array to the console

```
int scores[10];
int i;

for(i= 0; i < 10; i++){
    printf(“%i”, scores[ i ]);
}
```

Arrays

- Arrays in C
  - Mbed example
    - Read an input pin 5 times and store the results in an array

```
int pin_inputs[5];
int i;

for(i= 0; i < 5; i++){
    pin_inputs[i] = MyPin.read();
    wait_us(T_WAIT);
}
```

Arrays

- Arrays in C
  - Assignment
    - Whole arrays cannot be used on the right side of an assignment operator

```
int array1[10];
int array2[10];
…
~~array2 = array1;~~
```

Arrays

- Arrays in C
  - Assignment
    - Arrays must be copied element by element

```
int array1[7];
int array2[7];
int i;
…
for(i = 0; i < 7; i++){
   array2[ i ] = array1[ i ];
}
```

Arrays

- Arrays in C
  - Example
    - Exchange the values in array1 and array2

Arrays

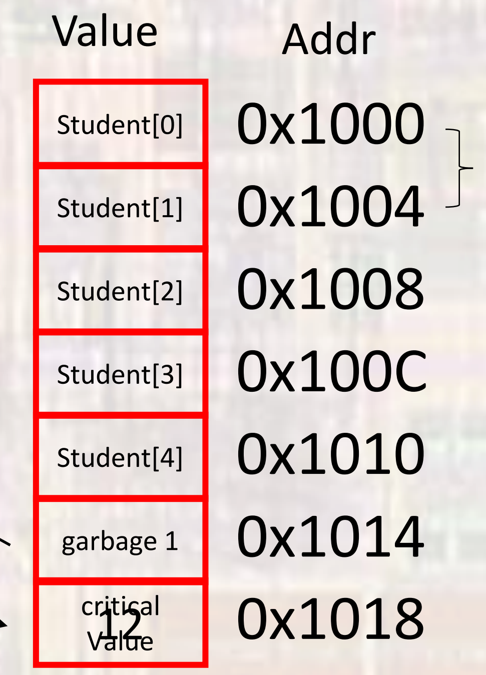
- Arrays in C
  - Example
    - Exchange the values in array1 and array2

```
int array1[10];
int array2[10];
int i;
…
for(i = 0; i < 10; i++){
   int tmp;
   tmp = array1[ i ];
   array1[ i ] = array2[ i ];
   array2[ i ] = tmp;
}
```

Arrays

- Index Range Checking
  - C does NOT check array index ranges

```
int Student[5];
…

foo = Student[5];
    sets foo = garbage

Student[6] = 12;
    overwrites critical data value
```



Arrays

- Passing array values
  - Passing array values works just like any other value

```
void fun1 (int zoo);
void fun2 (float* soo);

fun1(foo);                // passes the value of foo to function
                          // fun1
fun1(myArray[3]);         // passes the value of myArray[3]
                          // to function fun1

fun2(&boo);               // passes a pointer to boo (the address)
                          // to function fun2
fun2(&myFloatArray[3]);   // passes a pointer to myFloatArray
                          // element 3 (the address)
                          // to function fun2
```

Arrays

- Passing array values
  - Passing the whole array
    - If we pass all the elements of a large array to multiple functions, we use up a lot of data memory
    - Instead, we pass the address of the array (by reference)
  - Remember – the name of the array is already the address of the beginning of the array

declaration

```
void fun3(int ary[ ]);        // the array notation name[]
                              // tells the compiler it is expecting an
                              // address
…
```

call

```
fun3(myArray);                // the array name is already an
                              // address
```

Arrays

- Passing array values
  - Passing the whole array
    - To make our functions more useful we will usually pass the whole array AND the number of elements

declaration

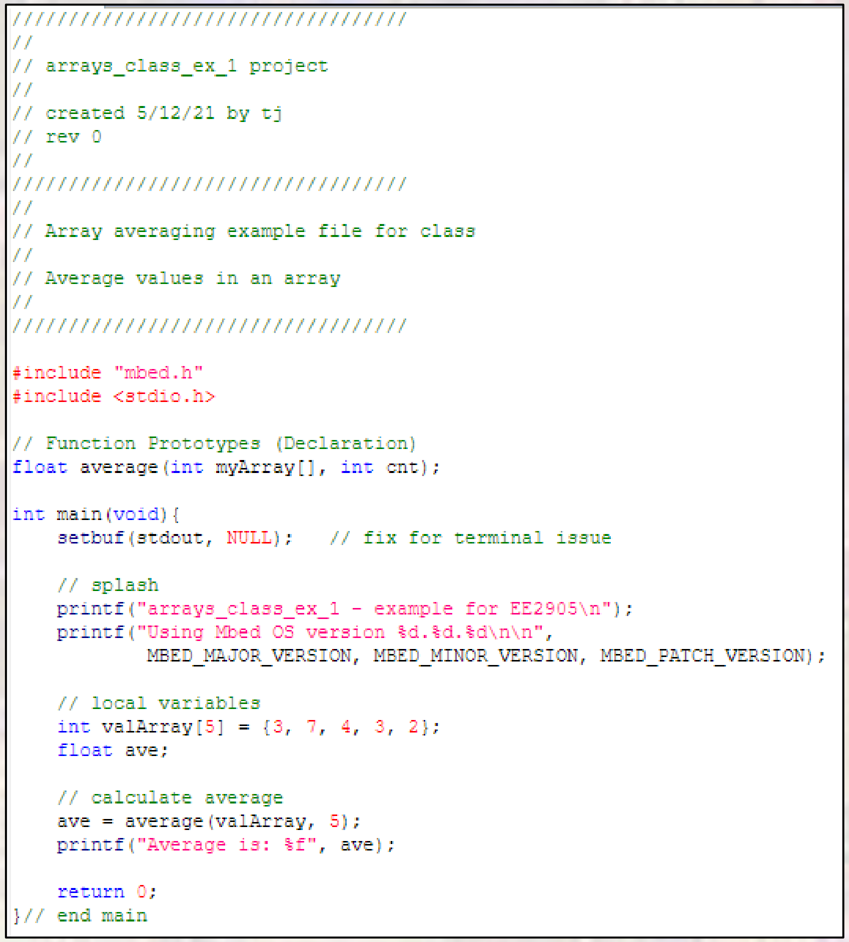
```
void fun3(int ary[ ], int n);  // the array notation name[]
                               // tells the compiler it is expecting an
                               // address
…
fun3(myArray, 10);             // the array name is already an
                               // address
```

call

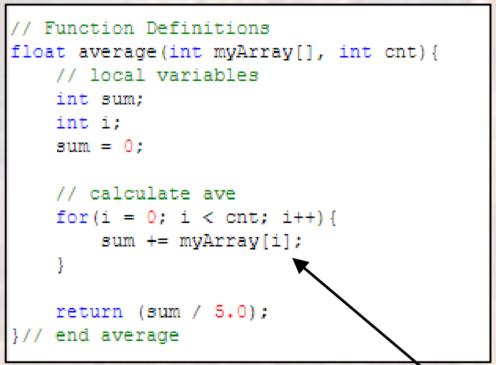
Arrays

- Passing array values
  - Array average program

This function works for any size array



```
////////////////////////////////////
//
// arrays_class_ex_1 project
//
// created 5/12/21 by tj
// rev 0
//
////////////////////////////////////
//
// Array averaging example file for class
//
// Average values in an array
//
////////////////////////////////////

#include "mbed.h"
#include <stdio.h>

// Function Prototypes (Declaration)
float average(int myArray[], int cnt);

int main(void){
    setbuf(stdout, NULL);    // fix for terminal issue

    // splash
    printf("arrays_class_ex_1 - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
            MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // local variables
    int valArray[5] = {3, 7, 4, 3, 2};
    float ave;

    // calculate average
    ave = average(valArray, 5);
    printf("Average is: %f", ave);

    return 0;
}// end main
```



```
// Function Definitions
float average(int myArray[], int cnt){
    // local variables
    int sum;
    int i;
    sum = 0;

    // calculate ave
    for(i = 0; i < cnt; i++){
        sum += myArray[i];
    }

    return (sum / 5.0);
}// end average
```

remember the index is an offset from the beginning of the array

Arrays

- Passing array values
  - What if we want to pass the whole array to a function but we do not want the function to modify the array?
  - Declare the array as a constant in the function declaration and definition

```
float average(int myArray[ ], int n);                // modifiable
→
float average(const int myArray[ ], int n);          // non-modifiable
```