

Arrays

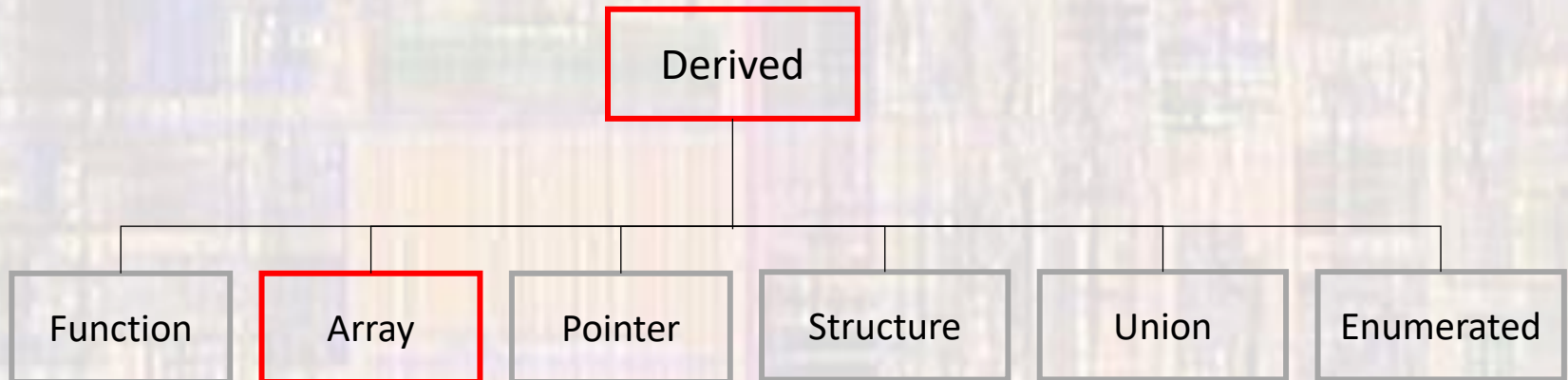
Last updated 12/23/20

Arrays

- These slides introduce arrays
- Upon completion: You should be able to interpret and code using arrays

Arrays

- C Types
 - Arrays are a Derived type



Arrays

- Arrays
 - Grouping of similar items

Student 0

Student 1

Student 2

Student 3

Student 4

Student₀

Student₁

Student₂

Student₃

Student₄

Student[0]

Student[1]

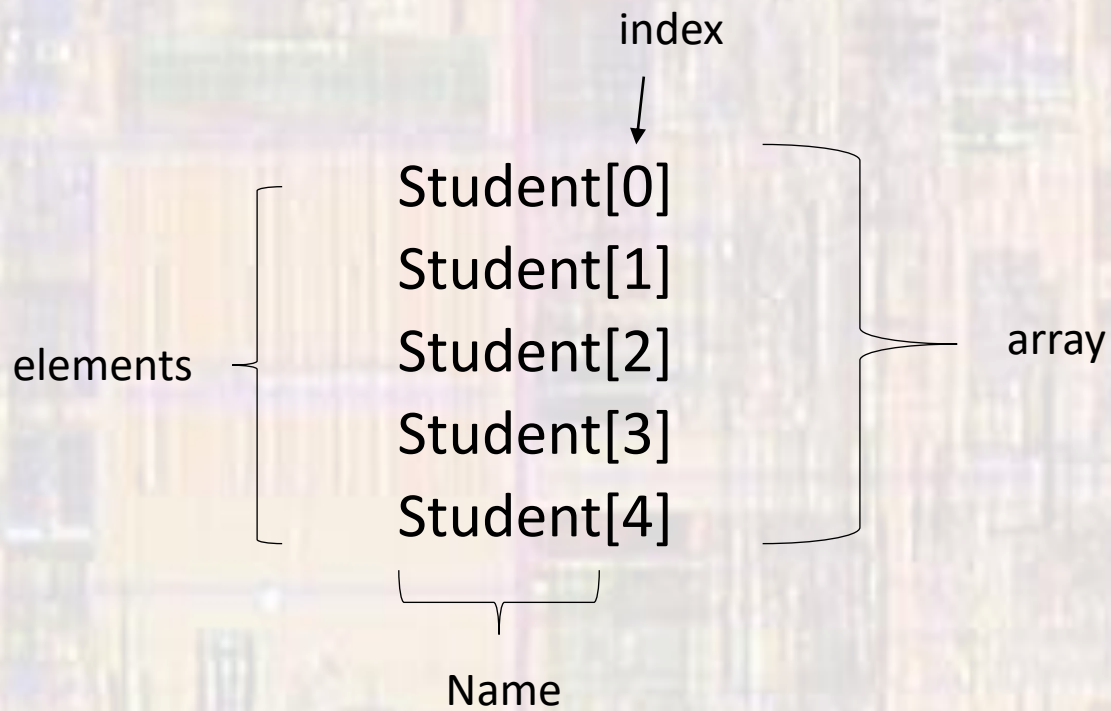
Student[2]

Student[3]

Student[4]

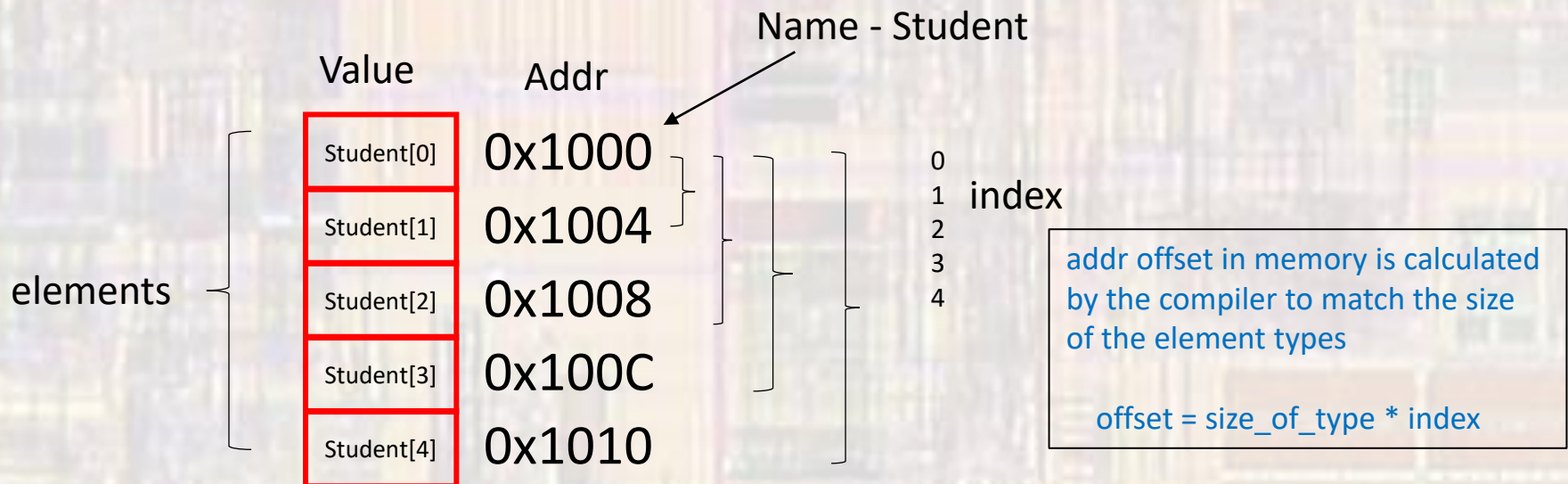
Arrays

- Array notation



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



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

these are complex – and we will not use them in EE1910

Arrays

- Arrays in C

Initialization

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

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

element 0



myArray

Arrays

- Arrays in C

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
```

5	4	0	0	0
---	---	---	---	---

// others are set to 0

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

5	4	3	2	1
---	---	---	---	---

// initialization values

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

0	0	0	0	0
---	---	---	---	---

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("%d", &scores[ i ]);  
}
```

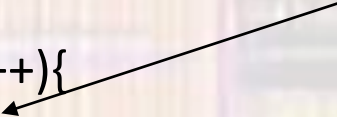
Arrays

- 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("%d", &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("%d", scores[ i ]);  
}
```

Arrays

- Arrays in C
 - MSP example
 - Read an input pin 5 times and store the result in an array

```
int pin_input[5];
int i;

for(i= 0; i < 5; i++){
    pin_input[i] = (P6->IN & 0x08) && 0x01;
    __delay_cycles(3000000);
}
```

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
```

Value	Addr
Student[0]	0x1000
Student[1]	0x1004
Student[2]	0x1008
Student[3]	0x100C
Student[4]	0x1010
garbage 1	0x1014
critical Value	0x1018

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
 - Array average program

```
/*
 * array_ave.c
 *
 * Created on: Dec 18, 2020
 * Author: johnsontimoi
 */
////////////////////////////////////
//
// array passing example for class
//
////////////////////////////////////
#include <stdio.h>

// Function Prototypes (declaration)
float average(int myArray[ ]);

int main(void){
    setbuf(stdout, NULL); // disable buffering

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

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

    return 0;
} // end main
```

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

    for(i = 0; i < 5; 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[ ]);           // modifiable
```

→

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