More Pointers

Last updated 10/29/20

More Pointers

- These slides discuss pointers and arrays
- Upon completion: You should be able interpret and code using array pointers

More Pointers

- Pointer Review

  - Declaration

    ```
    int* myIntPtr;          // define a ptr to a variable of type int
    ```

  - Definition

    ```
    myIntPtr = &foo;        // set myIntPtr to the address of foo
    ```

  - Dereference

    ```
    boo = *myIntPtr;        // boo equals value in memory
                            // location pointed to by
                            // myIntPtr
    ```

More Pointers

- Pointer Review

  - Passing an address to a function

```
func1(foo, &boo);            // pass the value of foo to the fn
                             // pass the address of boo to the fn
```

  - Expect an address in a function

```
void func1(int soo, int* zoo){
                             // read an int and call it soo locally
                             // read a pointer (address) to a
                             // variable of type int and call it
                             // zoo locally
```

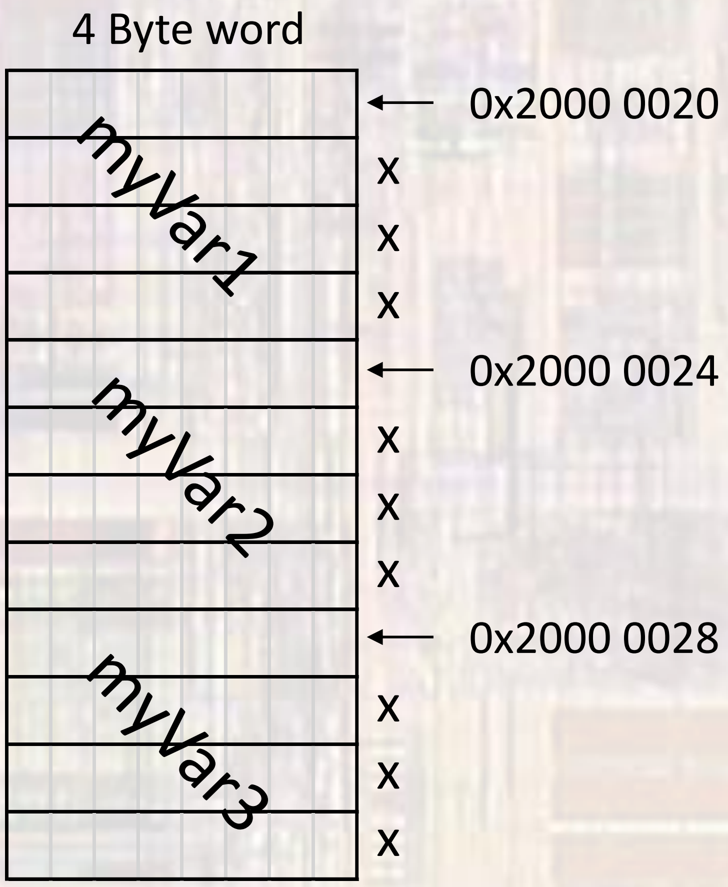
  - Use an address in a function

```
*zoo += soo;                 // add the value pointed to by the
                             // pointer zoo to the value of soo
                             // and put it back into the value
                             // of the variable pointed to by zoo
```

More Pointers

- Pointers in memory

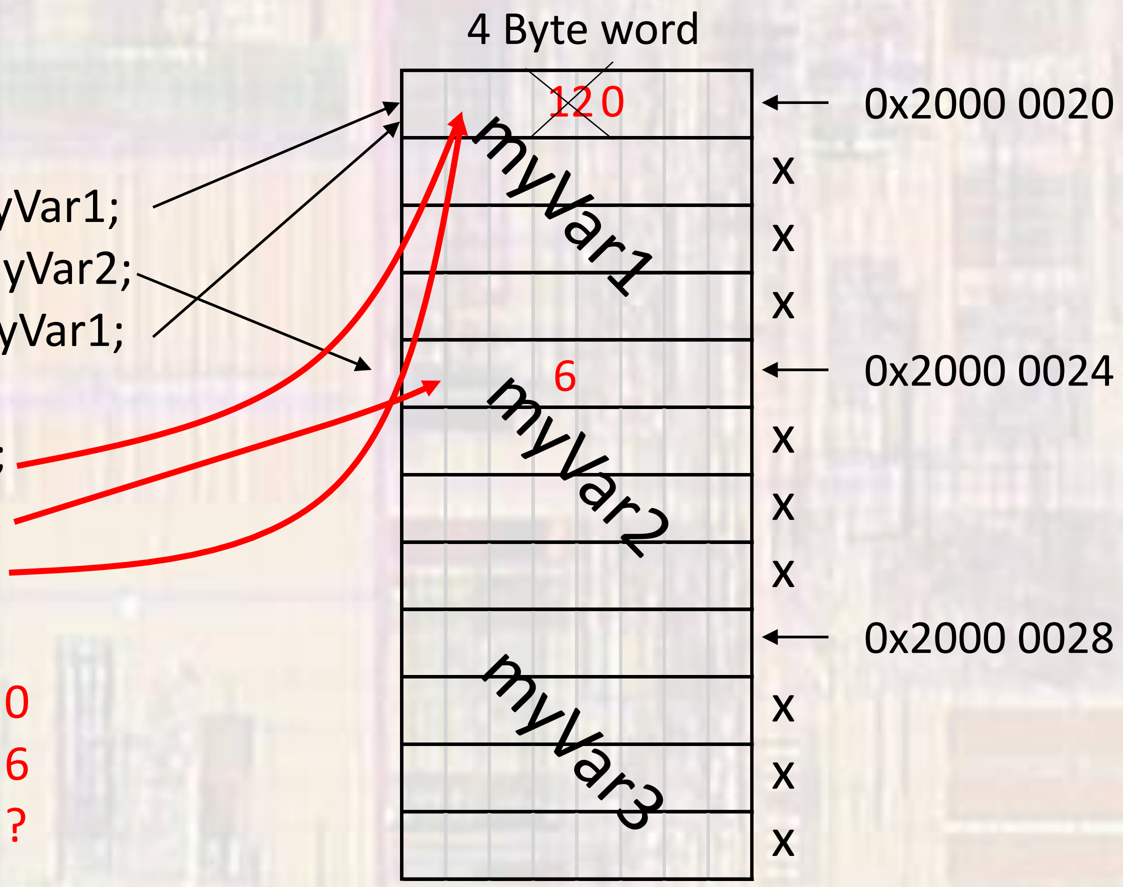
```
int* foo;
int* boo;
int* soo;

foo = &myVar1;
boo = &myVar2;
soo = &myVar1;

*foo = 12;
*boo = 6;
*soo = 0;

myVar1 =
myVar2 =
myVar3 =
```



More Pointers

- Pointers in memory

```
int* foo;
int* boo;
int* soo;

foo = &myVar1;
boo = &myVar2;
soo = &myVar1;

*foo = 12;
*boo = 6;
*soo = 0;

myVar1 = 0
myVar2 = 6
myVar3 = ?
```



More Pointers

- Pointer Arithmetic
  - Pointers have a type
  - The type can be used to allow pointer arithmetic
    - Addition and subtraction of pointers is done in increments of the “type” size.
      - E.g. ints → 4Bytes, chars → 1Byte
      - The allowed operations on pointers are: +, -, ++, --

```
int* foo;
int* soo;
foo = &boo;          // assume boo is located at 0x1000 with value 25
soo = foo + 2;       // soo now has the value 0x1008
foo++;               // foo now has the value 0x1004
loo = *(soo – 2);    // loo now equals 25
```

More Pointers

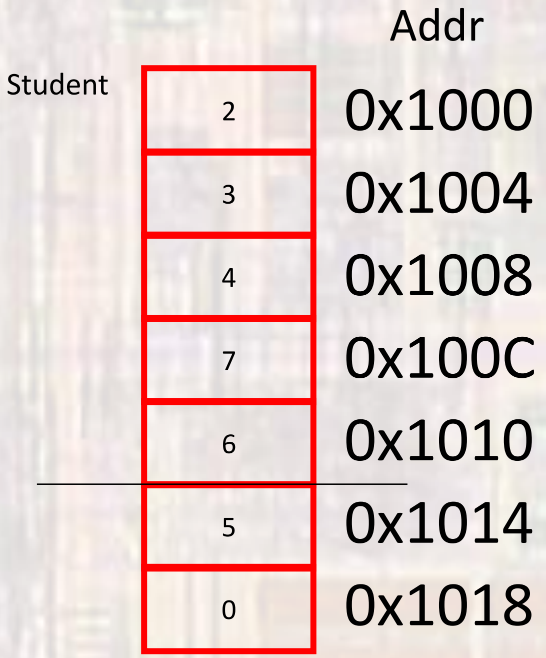
- Pointers and Arrays
  - Reminder: the name of an array is actually a pointer to the 0th element of the array

```
int myArray[ ];       // myArray holds the value 0x1000 (ptr)

myArray + 2           evaluates to 0x1008 (ptr arithmetic)

*(myArray + n)  is equivalent to myArray[n]
       ↑
       |
pointer arithmetic
```

More Pointers

- Pointers and Arrays

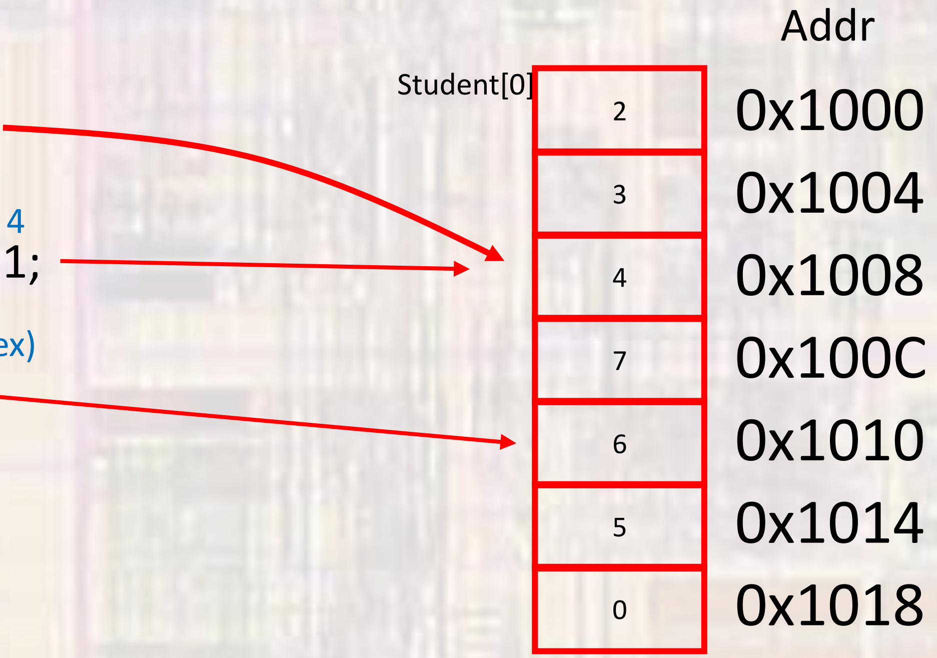
```
int Student[5];
int* myPtrA;
int* myPtrB;
int* myPtrC;

myPtrA = &Student[2];

myPtrB = &Student[1] + 1;

myPtrC = Student

*myPtrA
*myPtrB
*(myPtrC + 2)
*(Student + 2)
```



More Pointers

- Pointers and Arrays

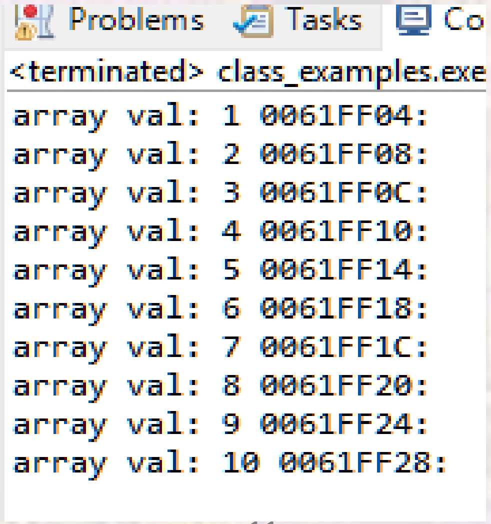
```
int Student[5];
int* myPtrA;
int* myPtrB;
int* myPtrC;

  1008       1008
myPtrA = &Student[2];

  1008          1004        +  4
myPtrB = &Student[1] + 1;

  1010        1000      +  10 (hex)
myPtrC = Student + 4;

*myPtrA              4
*myPtrB              4
*(myPtrC - 2)        4
*(Student + 2)       4
```



More Pointers

- Pointers and Arrays

```
//  Local variables
  int myArray[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  //   print array
  int i;
  for(i = 0; i < 10; i = i + 1){
      printf("array val: %i %p: \n", *(myArray + i), myArray + i);
  }
```

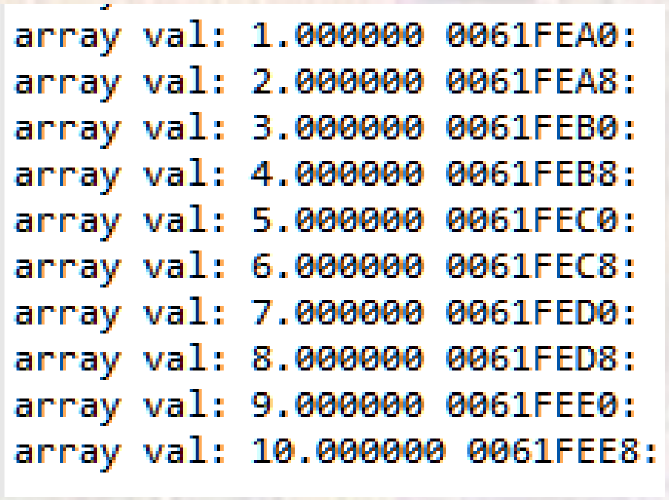


```
<terminated> class_examples.exe
array val: 1 0061FF04:
array val: 2 0061FF08:
array val: 3 0061FF0C:
array val: 4 0061FF10:
array val: 5 0061FF14:
array val: 6 0061FF18:
array val: 7 0061FF1C:
array val: 8 0061FF20:
array val: 9 0061FF24:
array val: 10 0061FF28:
```

More Pointers

- Pointers and Arrays

```
// Local variables
 double myArray2[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

 //  print array
 int i;
 for(i = 0; i < 10; i = i + 1){
    printf("array val: %f %p: \n", *(myArray2 + i), myArray2 + i);
 }
```

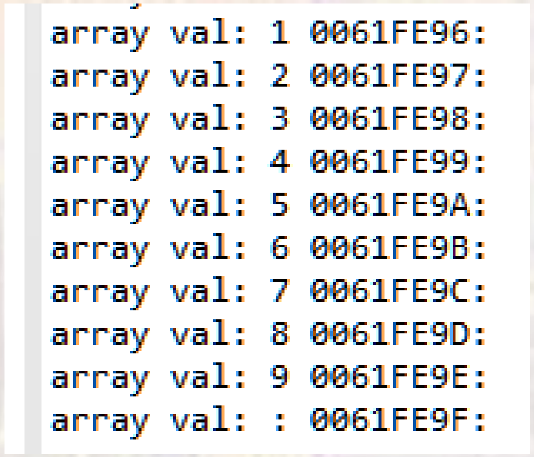


```
array val: 1.000000 0061FEA0:
array val: 2.000000 0061FEA8:
array val: 3.000000 0061FEB0:
array val: 4.000000 0061FEB8:
array val: 5.000000 0061FEC0:
array val: 6.000000 0061FEC8:
array val: 7.000000 0061FED0:
array val: 8.000000 0061FED8:
array val: 9.000000 0061FEE0:
array val: 10.000000 0061FEE8:
```

More Pointers

- Pointers and Arrays

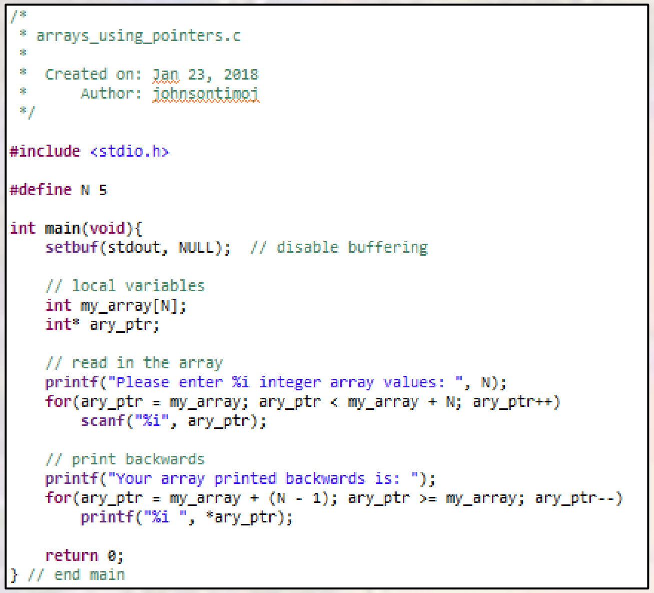
```
// Local variables
char myArray3[10] = {49, 50, 51, 52, 53, 54, 55, 56, 57, 58 };

//  print array
int i;
for(i = 0; i < 10; i = i + 1){
   printf("array val: %c %p: \n", *(myArray3 + i), myArray3 + i);
}
```



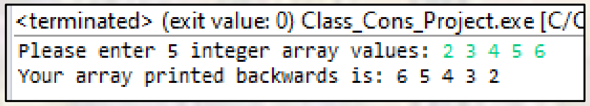
```
array val: 1 0061FE96:
array val: 2 0061FE97:
array val: 3 0061FE98:
array val: 4 0061FE99:
array val: 5 0061FE9A:
array val: 6 0061FE9B:
array val: 7 0061FE9C:
array val: 8 0061FE9D:
array val: 9 0061FE9E:
array val: : 0061FE9F:
```

More Pointers

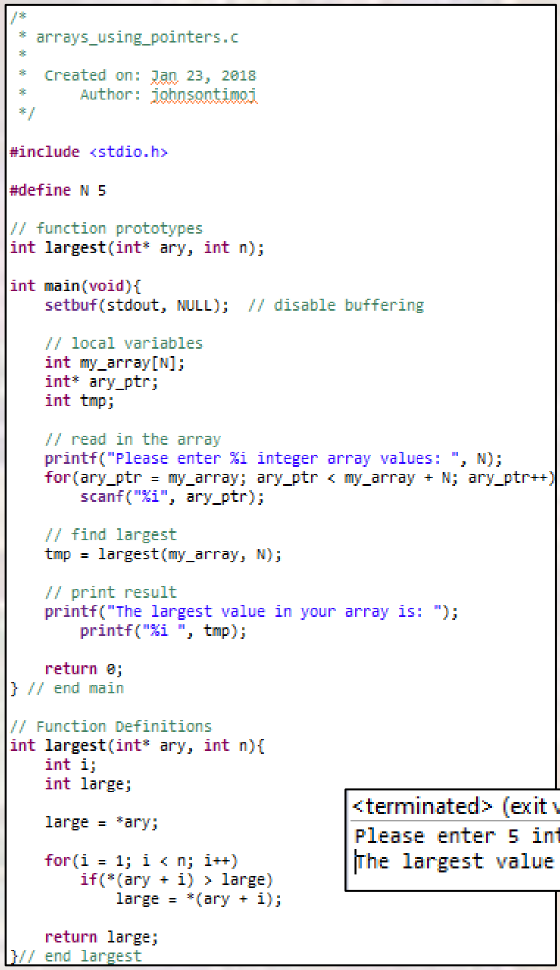
- Pointers and Arrays
    - The pointer terminology can replace our array terminology



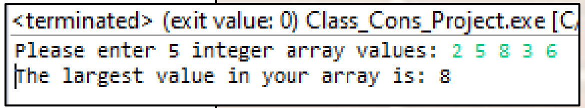
```c
/*
 * arrays_using_pointers.c
 *
 *  Created on: Jan 23, 2018
 *      Author: johnsontimoj
 */

#include <stdio.h>

#define N 5

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

    // local variables
    int my_array[N];
    int* ary_ptr;

    // read in the array
    printf("Please enter %i integer array values: ", N);
    for(ary_ptr = my_array; ary_ptr < my_array + N; ary_ptr++)
        scanf("%i", ary_ptr);

    // print backwards
    printf("Your array printed backwards is: ");
    for(ary_ptr = my_array + (N - 1); ary_ptr >= my_array; ary_ptr--)
        printf("%i ", *ary_ptr);

    return 0;
} // end main
```



```
<terminated> (exit value: 0) Class_Cons_Project.exe [C/C
Please enter 5 integer array values: 2 3 4 5 6
Your array printed backwards is: 6 5 4 3 2
```



```c
/*
 * arrays_using_pointers.c
 *
 *  Created on: Jan 23, 2018
 *      Author: johnsontimoj
 */

#include <stdio.h>

#define N 5

// function prototypes
int largest(int* ary, int n);

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

    // local variables
    int my_array[N];
    int* ary_ptr;
    int tmp;

    // read in the array
    printf("Please enter %i integer array values: ", N);
    for(ary_ptr = my_array; ary_ptr < my_array + N; ary_ptr++)
        scanf("%i", ary_ptr);

    // find largest
    tmp = largest(my_array, N);

    // print result
    printf("The largest value in your array is: ");
        printf("%i ", tmp);

    return 0;
} // end main

// Function Definitions
int largest(int* ary, int n){
    int i;
    int large;

    large = *ary;

    for(i = 1; i < n; i++)
        if(*(ary + i) > large)
            large = *(ary + i);

    return large;
}// end largest
```



```
<terminated> (exit value: 0) Class_Cons_Project.exe [C/
Please enter 5 integer array values: 2 5 8 3 6
The largest value in your array is: 8
```