

# COP 3223H: Introduction to C Programming

Fall 2023

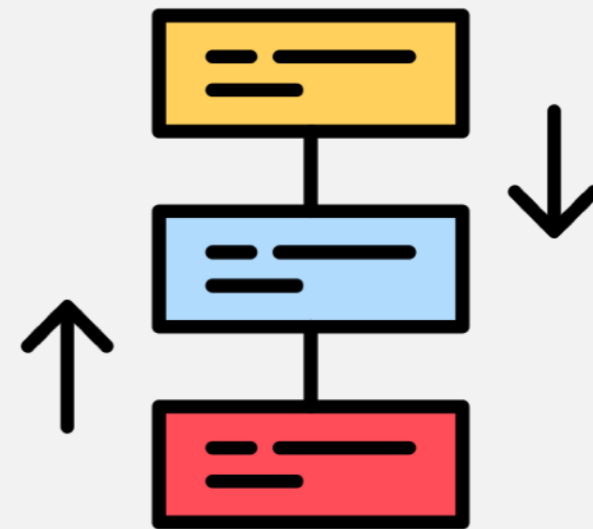


University of  
Central Florida

---

Dr. Kevin Moran

## *Week 8- Class II:* Arrays Part I





- *Large Programming Assignment 1* is due on Friday!!
- SPA 1 Grades out now.
- SPA 2 Grades out today.

# Today's Agenda



1. File I/O Example
2. Begin Discussing Arrays in C

# Quick Review





- In C we can access files (such as text files)
- This access allows for reading and writing.
  - Reading – Input
  - Writing – Output
- There is a special kind of variable in C that allows us access for text files.
- *File Pointers!*

```
FILE *inp; // pointer to input file  
FILE *outp; // pointer to output file
```



- There are two basic types of access we will learn in this class
  - Reading – this allows the program to collect input from a text file. Think of it like scanf for collecting input from the keyboard
  - Writing – this allows the program to write output to a text file. Think of it like printf for displaying output to the monitor

# Other Types of File I/O Access



- There are other modes for FILE I/O Access besides r and w mode.
  - *a – append mode*
    - Adds content to the next available space in the File
  - *r+ – both reading and writing*
    - Acts as both r and w mode. Assumes that File exists in memory
    - If file does not exist then it doesn't work
  - *w+ – both reading and writing*
    - Acts as both r and w mode. Doesn't assume that File exist in memory
    - If it does exist already, content will be deleted by setting the length to zero bytes
    - If it doesn't exist, it will create the File
  - *a+ – both reading and writing*
    - If file doesn't exist, it will create it
    - When reading, pointer starts at the beginning of the file content
    - Writing to file will only be appended

# Syntax for File Reading/Writing



```
// preparing files for input and output  
inp = fopen("indata.txt", "r");  
outp = fopen("outdata.txt", "w");
```

```
fscanf(inp, "%lf", &item); // reading file  
fprintf(outp, "%f", item); // writing file
```



# printf, scanf, fprintf, and fscanf



```
FILE *inp; // pointer to input file  
FILE *outp; // pointer to output file
```

```
// preparing files for input and output  
inp = fopen("indata.txt", "r");  
outp = fopen("outdata.txt", "w");
```

```
{ scanf("%lf", &item); // reading input from command line  
  fscanf(inp, "%lf", &item); // reading input from file
```

```
{ printf("%f", item); // printing information to command line  
  fprintf(outp, "%f", item); // writing file
```

```
fclose(inp);  
fclose(outp);
```

Notice the  
placeholder  
and variable  
address

Notice the  
placeholder  
and variable

The only  
addition is the  
file pointer!

# The `fscanf` Function



- Remember way back we talked about what the `scanf` function returns?
  - An integer value representing the number of values successfully processed.
- We just observed the similar syntax for the `scanf` and `fscanf` functions.
- Does `fscanf` return a similar value?
  - YES!!
  - It returns the number of values processed successfully. This also includes 0 if it was unable to process the first value being collected.

# EOF Macro Constant



- C has a special *predefined* macro constant called EOF in the stdio header file.
- EOF stands for “End Of File”
  - The value of EOF is  $-1$ . 0 is still used if it can read something potential, BUT wasn't processed successfully.
- EOF is widely used to assist with reading an ENTIRE file.

```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, "%lf", &item) != EOF){
    printf("item = %d\n", item);
}

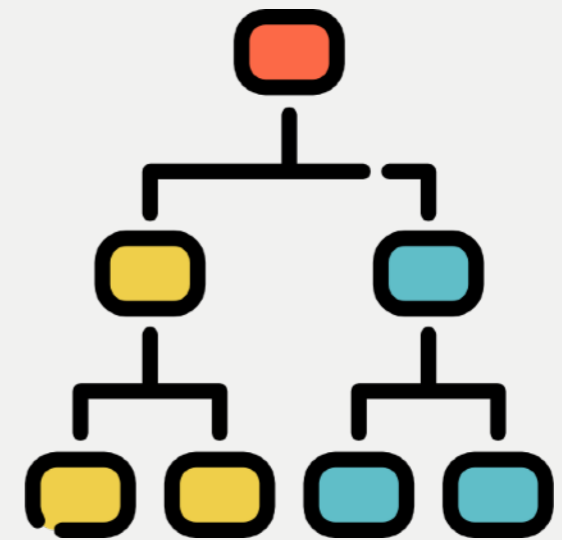
fclose(inp);
```

# Arrays



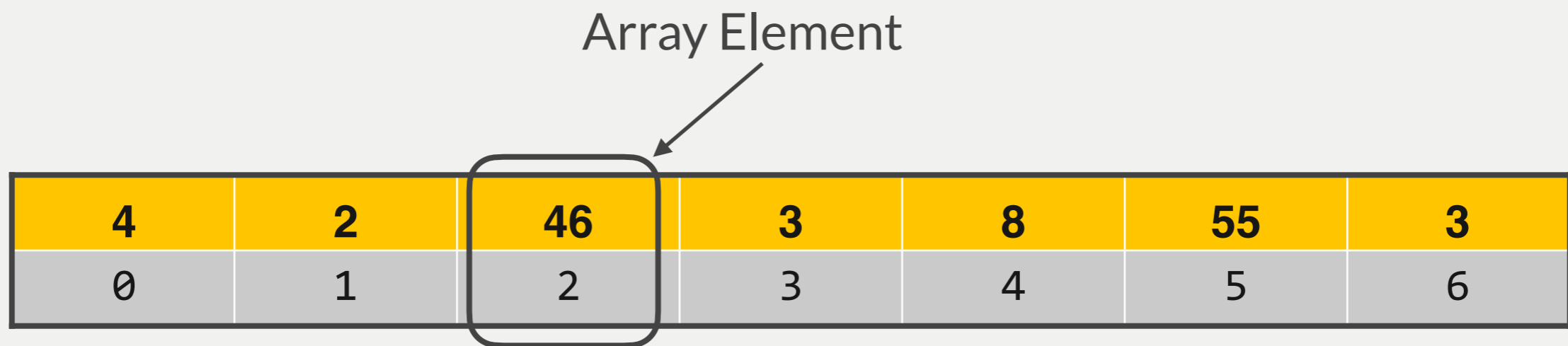


- Data structures is a composite of related data items stored under the same name.
- Data structures allows programmers to store data in a more organized fashion.





- An Array is a collection of data items of the same type.
- An array element is a data item that is part of an array.
- An array is a collection of two or more adjacent memory cells.



# Declaring an Array



```
int x[8];
```

Type of values  
stored in array

Identifier

Number of  
elements

# Arrays and Stack Visualization



```
int x[8];
```

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	



# Arrays and Stack Visualization



```
int x[8];
```

Here we have an array (called x) of 8 elements. That means there are 8 adjacent cells occupied.

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	

# Arrays and Stack Visualization



```
int x[8];
```

Here we have an array (called x) of 8 elements. That means there are 8 adjacent cells occupied.

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	

# Array Demo

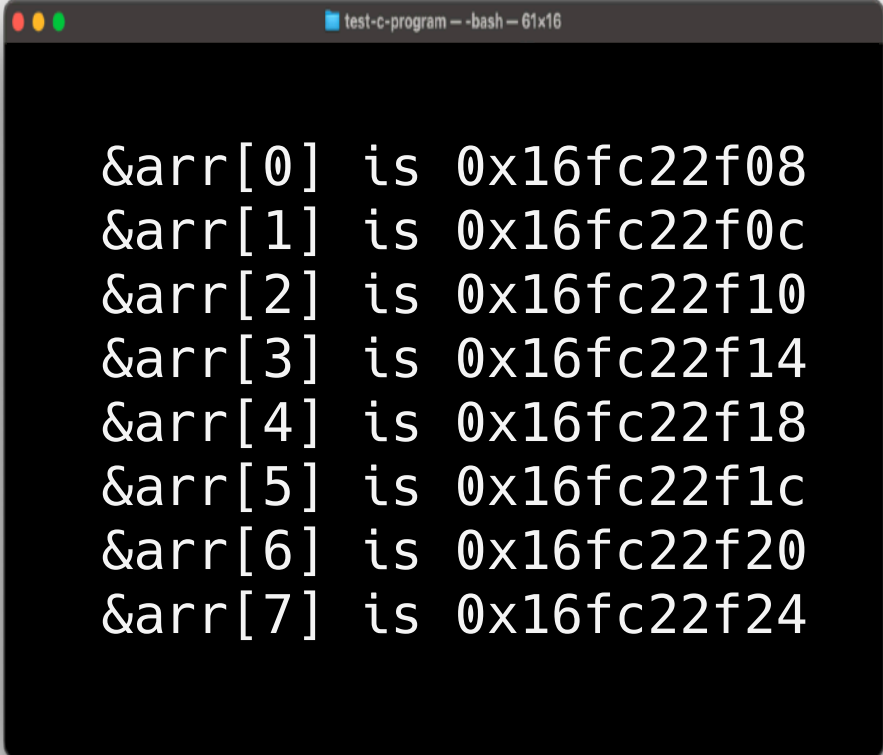


# Arrays and Stack Visualization



Addresses in Memory  
are hexadecimal

```
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```



```
&arr[0] is 0x16fc22f08  
&arr[1] is 0x16fc22f0c  
&arr[2] is 0x16fc22f10  
&arr[3] is 0x16fc22f14  
&arr[4] is 0x16fc22f18  
&arr[5] is 0x16fc22f1c  
&arr[6] is 0x16fc22f20  
&arr[7] is 0x16fc22f24
```

# Arrays and Stack Visualization



Addresses in Memory  
are hexadecimal

```
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```

```
test-c-program -- bash -- 61x16  
&arr[0] is 0x16fc22f08  
&arr[1] is 0x16fc22f0c  
&arr[2] is 0x16fc22f10  
&arr[3] is 0x16fc22f14  
&arr[4] is 0x16fc22f18  
&arr[5] is 0x16fc22f1c  
&arr[6] is 0x16fc22f20  
&arr[7] is 0x16fc22f24
```

Addresses are Increasing by 4 ...  
Why is this?

# Arrays and Stack Visualization



Addresses in Memory  
are hexadecimal

```
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```

```
test-c-program -- bash -- 61x16  
&arr[0] is 0x16fc22f08  
&arr[1] is 0x16fc22f0c  
&arr[2] is 0x16fc22f10  
&arr[3] is 0x16fc22f14  
&arr[4] is 0x16fc22f18  
&arr[5] is 0x16fc22f1c  
&arr[6] is 0x16fc22f20  
&arr[7] is 0x16fc22f24
```

Addresses are Increasing by 4 ...  
Why is this?

ints are 4 bytes!

# Arrays and Stack Visualization



Addresses in Memory  
are hexadecimal

```
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```

```
test-c-program -- bash -- 61x16  
&arr[0] is 0x16fc22f08  
&arr[1] is 0x16fc22f0c  
&arr[2] is 0x16fc22f10  
&arr[3] is 0x16fc22f14  
&arr[4] is 0x16fc22f18  
&arr[5] is 0x16fc22f1c  
&arr[6] is 0x16fc22f20  
&arr[7] is 0x16fc22f24
```

Addresses are Increasing by 4 ...  
Why is this?

ints are 4 bytes!

What happens for doubles/floats?



- We have already observed the adjacent memory cells by displaying their addresses.
- What about the actual array variables?
- For example: Where do you think the variable `arr` itself is located?

```
int arr[8];  
printf("&arr = %p\n", &arr);  
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```





- We have already observed the adjacent memory cells by displaying their addresses.
- What about the actual array variables?
- For example: Where do you think the variable `arr` itself is located?

```
int arr[8];  
printf("&arr = %p\n", &arr);  
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```

```
&arr = 0x16dd72f08  
&arr[0] is 0x16dd72f08  
&arr[1] is 0x16dd72f0c  
&arr[2] is 0x16dd72f10  
&arr[3] is 0x16dd72f14  
&arr[4] is 0x16dd72f18  
&arr[5] is 0x16dd72f1c  
&arr[6] is 0x16dd72f20  
&arr[7] is 0x16dd72f24
```

# Array Variables



- We have already observed the adjacent memory cells by displaying their addresses.
- What about the actual array variables?
- For example: Where do you think the variable `arr` itself is located?

```
int arr[8];  
printf("&arr = %p\n", &arr);  
for(int x = 0; x < 8; x++){  
    printf("&arr[%d] is %p\n", x, &arr[x]);  
}
```

The first adjacent stack cell is the actual place where the array variable is stored in memory.

```
test-c-program --bash -- 61x16  
  
&arr = 0x16dd72f08  
&arr[0] is 0x16dd72f08  
&arr[1] is 0x16dd72f0c  
&arr[2] is 0x16dd72f10  
&arr[3] is 0x16dd72f14  
&arr[4] is 0x16dd72f18  
&arr[5] is 0x16dd72f1c  
&arr[6] is 0x16dd72f20  
&arr[7] is 0x16dd72f24
```



Slides adapted from Dr. Andrew Steinberg's  
COP 3223H course