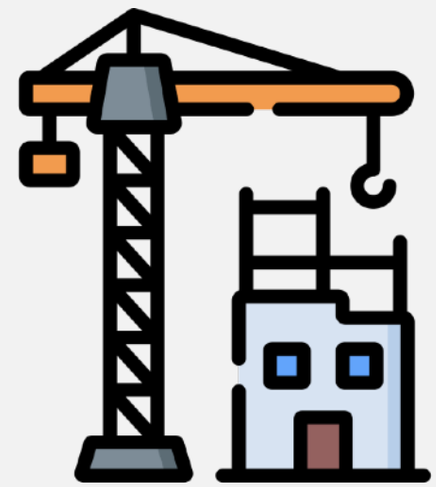
COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 12- Class 1: Structs Part I



Administrivia

- SPA 3 now due on Weds. - Python script out now
- SPA 4 and LPA 2 have been released, are be due on November, 10th, and November 17th respectively.
- Exam grades are on Webcourses!
- Mid-Semester Feedback Survey will be posted today.
  - Please complete to count as a quiz grade.
- No Class on Friday this week (Veterans Day)!

Today's Agenda

1. Intro to Structs

Review

`strlen` Example

```
char word[100] = "Mondays";
int len = strlen(word);
```

len = 7;

`strcpy` Example

```
char string1[8];
char string2[8] = "Cakes";

strcpy(string1, string2);
strcpy(string1, "Cookies");
```

strcat

```c
char string5[8] = "Vanilla";
char string6[8] = "Cookie";

strcat(string5, string6);

printf("string5 = %s\n", string5);
printf("string6 = %s\n", string6);
```

strcmp

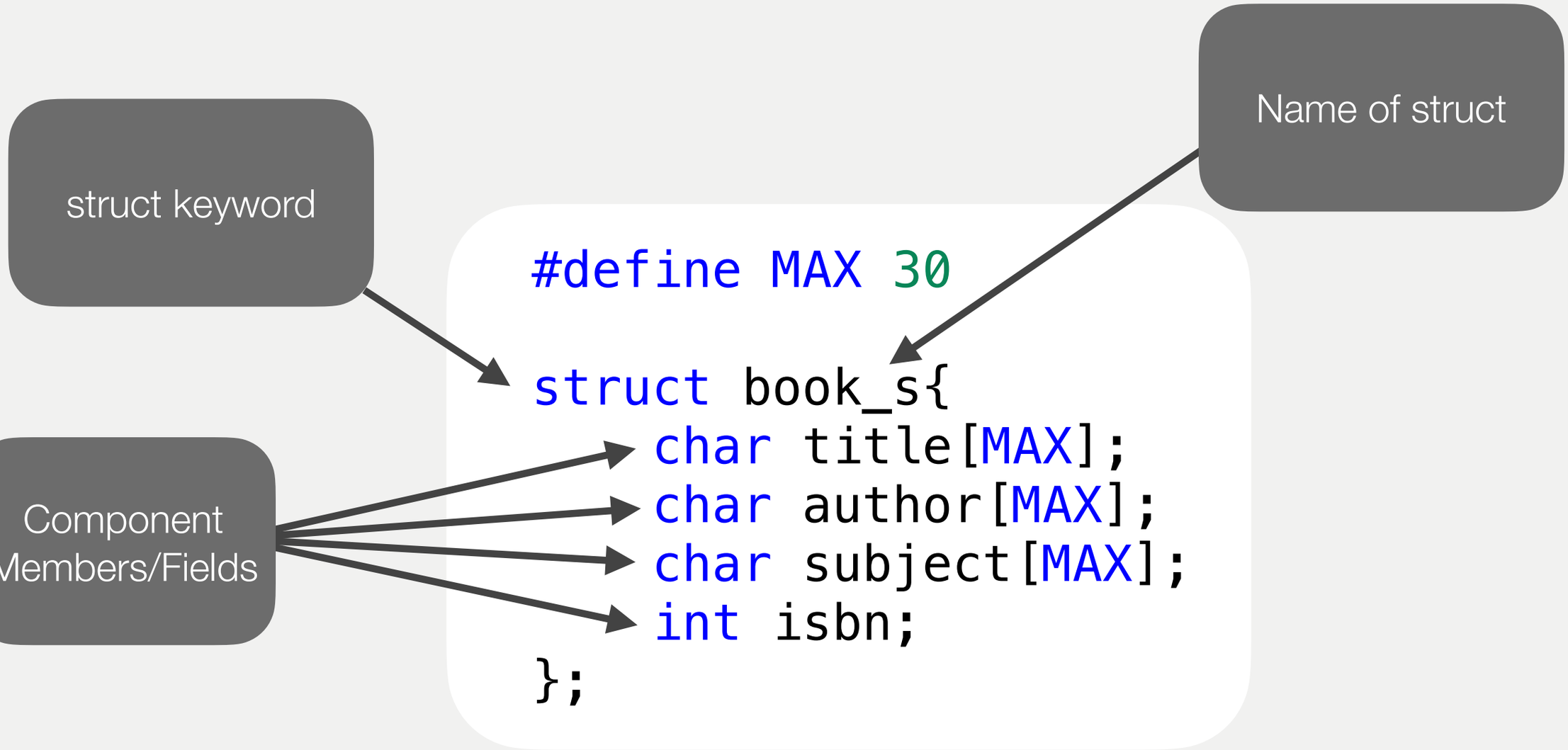
```
char string7[8] = "red";
char string8[8] = "blue";

int result = strcmp(string7, string8);

printf("result = %d\n", result);
```

Structs

User Defined Structure Types

- A database is a collection of information stored in a computer's memory or in a disk file.
- A database is subdivided into records, which are a collection of information about one data object.
- The structure of the record is determined by the structure of the object's data type.
- C provides several ways to define structures.

User Defined Structure Syntax



Organizing User-defined `structs`

- The structure definition must be placed at the top of your C file. More specifically, it should be between your preprocessor statements and function prototypes.

```c
// preprocessor statements
#include<stdio.h>
#define MAX 30

struct book_s{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
};

// user defined function prototypes

int main(void){

return 0;

}

// user defined function definitions
```

Declaring a `struct` variable in C

- The structure definition must be placed at the top of your C file. More specifically, it should be between your preprocessor statements and function prototypes.

```c
// preprocessor statements
#include<stdio.h>
#define MAX 30

struct book_s{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
};

// user defined function prototypes

int main(void){

return 0;

}

// user defined function definitions
```

Declaring a `struct` variable in C

- Declaring a structure variable is 99.9% the same as declaring any other sort of variable we have seen in this course.
- The ONLY difference is that we must use the keyword `struct`.

```c
#define MAX 30

struct book_s{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
};

int main(void){

struct book_s mybook;

return 0;

}
```

Assigning Values to the Components of a `Struct`

- C has a simple operator called the direct selection operator (`.`).
- This allows us to properly access and assign values in the structure.

```
struct book_s mybook;

strcpy(mybook.title, "Julius Cesar");
strcpy(mybook.author, "William Shakespeare");
strcpy(mybook.title, "Play");
mybook.isbn = 1234;
```

.title	“Julius Caesar”
.author	“William Shakespeare”
.subject	“Play"
.isbn	1234

Precedence and Associativity of Operators

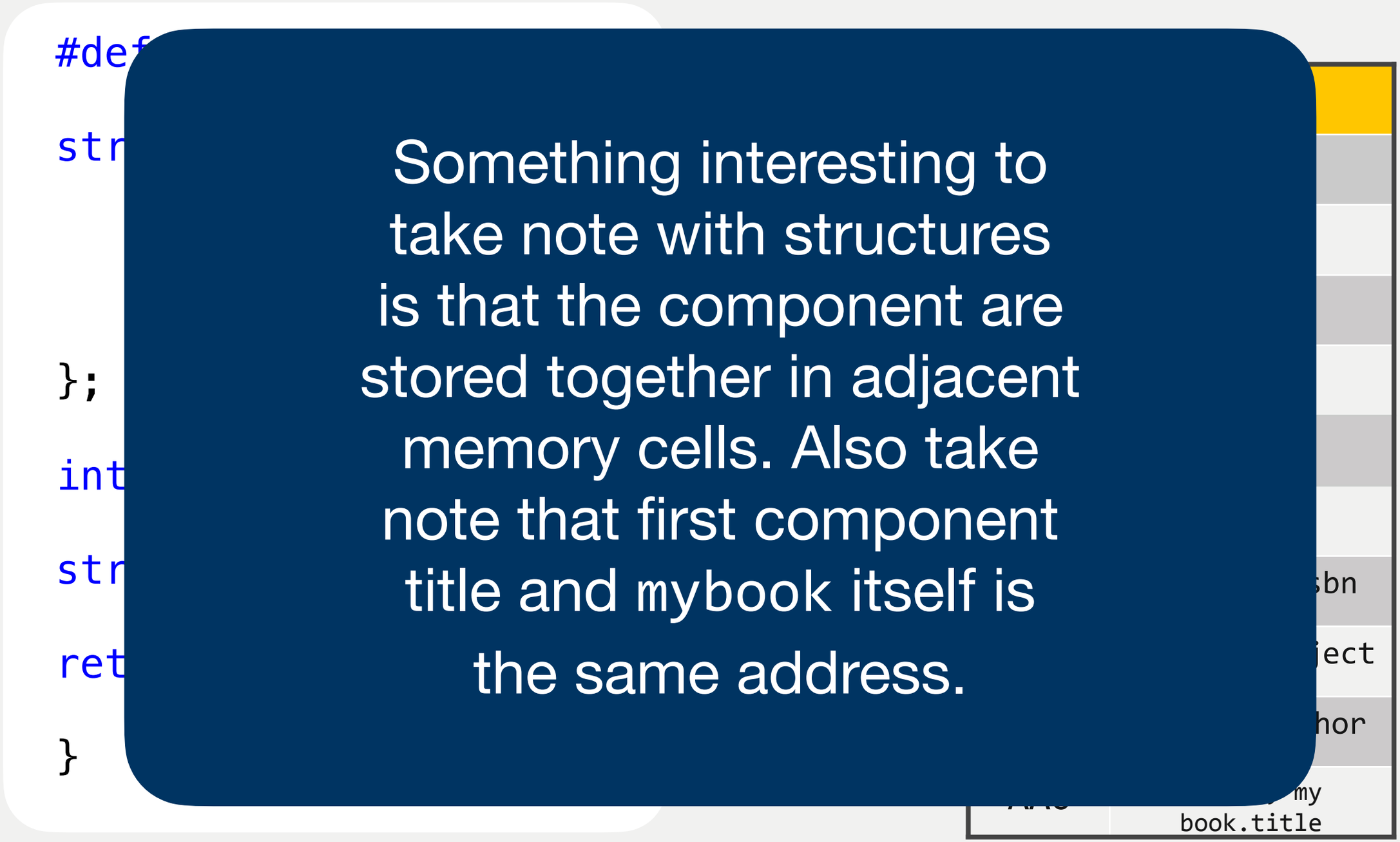
Precedence	Symbols	Operator	Associativity
Highest	`a[j] f(…)`	Subscripting, function calls, direct component selection	Left
	`++ —`	Postfix increment and decrement	Left
	`++ — ! - + & *`	Prefix increment and decrement, logical not, unary negation and plus, address of, indirection	Right
	`(type name)`	Casts	Right
	`* / %`	Multiplicative operators (multiplication, division, remainder)	Left
	`+ -`	Binary additive operators (addition and subtraction)	Left
	`< > <= >=`	Relational Operators	Left
	`!= =`	Equality/ Inequality Operators	Left
	`&&`	Logical And	Left
	`| |`	Logical Or	Left
Lowest	`+= = -= *= /= %=`	Assignment Operators	Right

Stack Space Visualization with `Structs`

```
#define MAX 30

struct book_s{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
};

int main(void){

struct book_s mybook;

return 0;

}
```

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	my book.isbn
AA2	mybook.subject
AA1	mybook.author
AA0	mybook, my book.title

Stack Space Visualization with `Structs`



Typedef Structures

- As you may of saw, every time we must use struct (such as a declaration), we are required to type out the keyword struct.

- C provides a special keyword that will allow programmers to avoid using the struct keyword.

- **`typedef`** is a special keyword that allows C to assign a name to some type.

```c
#define MAX 30
typedef struct{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
}book_t;

int main(void){

book_t mybook;

strcpy(mybook.title, "Julius Cesar");
strcpy(mybook.author, "William Shakespeare");
strcpy(mybook.title, "Play");
mybook.isbn = 1234;

return 0;

}
```

Hierarchical Structures

- Hierarchical structures are a structures containing components that are also structures.

```
typedef struct{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
}book_t;
```

```
typedef struct{

    char name[MAX];
    book_t mycollection[100];

}library_t;
```

Structs and Function Parameters

- Since structures are basically special variables, we can also use them as input/output parameters for user defined functions.
- We can perform both pass-by-value and pass-by-reference.
- With structures it is preferred that they are passed by reference since it easier (on the stack space) to pass the address (8 bytes always) of the struct rather than coping all the values of each component (number of bytes varies but could most likely be bigger than 8).

```
void displayBook(book_t mybook){
    printf("%s\n, mybook.title");
    printf("%s\n, mybook.author");
    printf("%s\n, mybook.subject");
    printf("%s\n, mybook.isbn");
}
```

Pass by Value

```
void displayBook(book_t *mybook){
    printf("%s\n, mybook.title");
    printf("%s\n, mybook.author");
    printf("%s\n, mybook.subject");
    printf("%s\n, mybook.isbn");
}
```

Pass by Reference

Indirect Component Selection Operator

- The indirect component selection operator is the character sequence -> placed between a pointer variable and a component name create a reference that follows the pointer to a structure and selects the component.

- While first one is valid to use, it can be a bit cumbersome to use, which is why C provides the indirect operator.

```
book_t *book_ptr = &mybook;

char title[MAX] = (*book_ptr).title;
char title2[MAX] = book_ptr->title;
```

Indirect Component Selection Operator

- The indirect component selection operator is the character sequence -> placed between a pointer variable and a component name create a reference that follows the pointer to a structure and selects the component.

- While first one is valid to use, it can be a bit cumbersome to use, which is why C provides the indirect operator.

```
book_t *book_ptr = &mybook;

char title[MAX] = (*book_ptr).title;
char title2[MAX] = book_ptr->title;
```

Some Common Struct Usages

Functions that Return `Structs`

```
book_t getBook(){
    book_t book;

    scanf("%s", book.title);
    scanf("%s", book.author);
    scanf("%s", book.subject);
    scanf("%d", book.isbn);

    return book;

}
```

Arrays of `Structs`

```
book_t book_array[100];
```

Arrays of `Structs`

```
book_t book_array[100];
```

Components	.title	.author	.subject	.isbn
mybook[0]	...	...	...	...
mybook[1]	...	...	...	...
mybook[2]	...	...	...	...
	...	...	...	...
mybook[99]	...	...	...	...

Traversing an Array of `Structs`

```
void displayLibrary(book_t mybook[]){
    for (int x = 0; x < 100; x++){
        printf("%s\n", mybook[x].title);
        printf("%s\n", mybook[x].author);
        printf("%s\n", mybook[x].subject);
        printf("%d\n", mybook[x].isbn);
    }
}
```

Demo

Acknowledgements

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