# COMP0204: Introduction to Programming for Robotics and AI

# Lecture 9: Structures in C

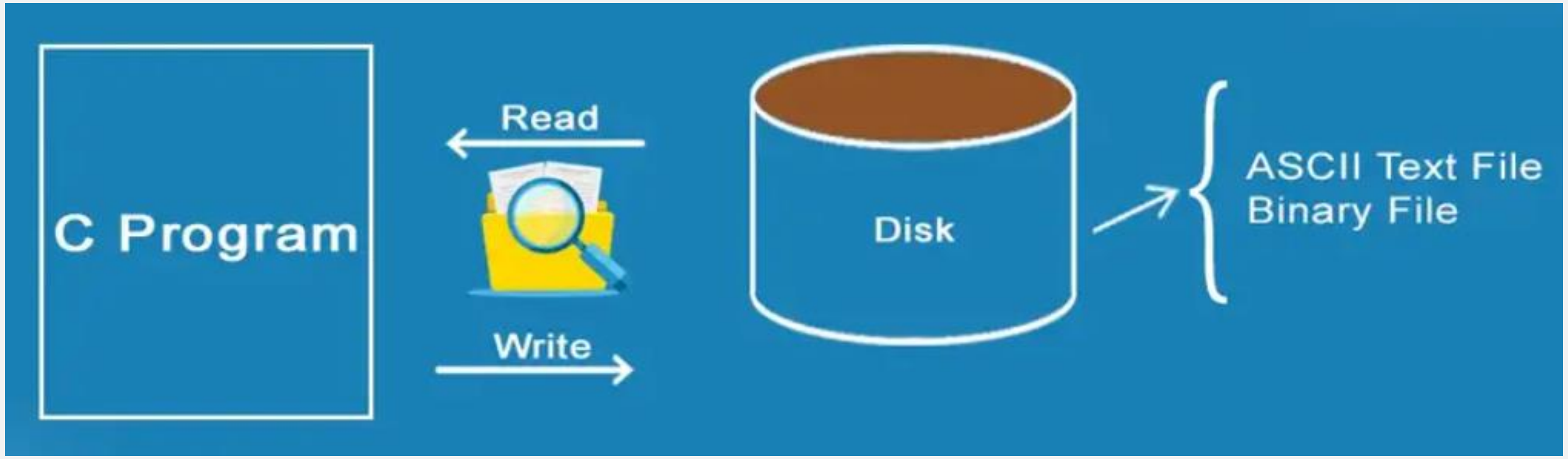
Course lead: Dr Sophia Bano

MEng Robotics and AI

UCL Computer Science

# Recap (previous week)

- What are the **FILES** in C programming?
- Why we **need** FILES
- Different **types** of FILES
- **Sequential** vs **direct** access FILES
- How we can **create/write/read** or close any FILE



- Assessment 6

# Today’s lecture

- Define a Structure in C
- Access Structure Members in C
- Structure Assignments in C
- Arrays of Structures in C
- Pass structures to functions in C
- Structure Pointers in C
- Dynamic Memory Allocation of Structure Type Variables

# Built-In Data Types (Primitive Data Types)

- Fundamental data types provided by the programming language.
- Integral to the language and are used to represent basic values.

**Example:** int, float, double, char, void

- These data types have **predefined characteristics**, such as size and behavior, determined by the programming language.
- They are efficient in terms of memory usage and execution speed.
- Operations on built-in data types are typically well-optimized by the compiler.

# User-Defined Data Types

- Created by the programmer to encapsulate and organize related data under a single name.
- These types are built using language constructs.

**Example: structures, typedef, unions, enumerations.**

- **Structures** - allow grouping different data types under a single name.
- **typedef** - used to create aliases for existing data types, improving code readability.
- Used to enhance code organization, maintainability, and expressiveness.

# Comparison

| Build-In Data Types | User-Defined Data Types |
|---|---|
| Predefined by the language, efficient, well-optimized by the compiler. | Created by the programmer to suit specific needs, provide abstraction and encapsulation. |
| Used for basic and fundamental data storage and manipulation. | Used for modeling complex entities, organizing related data, and improving code readability. |
| Fixed characteristics determined by the language. | Flexible, allowing the programmer to define the structure and behavior. |
| Represent basic values directly. | Provide a way to represent complex relationships and structures. |

# Structures

- **C arrays** allow you to define type of variables that can hold several data items of the **same kind** but **C structure** is a user-defined data type, which allows to combine data items of **different kinds**.

# Benefits of Structure

- Doesnot need multiple variable declaration.
- Good for data management and organization.

**Additional Benefits in Robot Programming:**

- Help in organizing information, from various components, sensors, and actuators, in a meaningful manner.
- Allows for abstraction. E.g: a robotic arm can be abstracted as a structure with properties like length, joints, etc.
- Provides a standardized way to represent data for communication, ensuring a consistent and organized exchange of information.

# Structures

- A collection of variables of different data types under a single name.
- A structure is defined to describe a group of related data, such as a "record" in a file.

e.g.

Student record (definition)

| ID Number | Family Name | Given Names | Date of Birth |
|---|---|---|---|

Example (content of such a record)

| | | | |
|---|---|---|---|
| 11112222 | " Andrew" | "John" | "12/04/1995" |

# Declaring Structures in C

- Defined using **struct** statement.
- <structName> is optional.
- Each <type> is a normal variable definition, such as int I; or float f; or any other valid variable definition.
- Before the final semicolon, we can specify one or more structure variables, but it is optional.

**Syntax :**

```
struct <structName>
{
     <type> <memberName1>;
     <type> <memberName2>;
     <type> <memberName3>;
      ......
} [one or more structure variables];
```

# Example : Declaring a C struct

```
struct Date
{
   int day;
   int month;
   int year;
};
```

structure **name**

**members** of the structure

(sometimes called "fields")

- This only declares a **new data type** called **Date**. You can then use it to create variables of type **Date**.
- **Important:** Date is not a variable. There is no memory allocated for it. It is merely a ***type*** (like int, float, etc).

# Defining a Structure Variable

Syntax :-

```
    <structName> <variableName>;
```

<u>Examples:</u>

```
Date birthday;
```

- creates a variable called **`birthday`** of type **`Date`**. This variable has 3 *components* (members) : **`day`**, **`month`**, and **`year`**.

```
Date today;
```

- creates another variable of type **`Date`**, also with component parts called **`day`**, **`month`** and **`year`**.

# Initializing Structure Variables

A structure variable can be defined and initialized at the same time as the structure is declared, but this is not recommended.

```
struct Date
{
    int day;
    int month;
    int year;
} today = {5, 8, 1996};
```

**today**

|  |  |  |
|---|---|---|
| 5 | 8 | 1996 |

*Preferable* method (easier to understand):

- Date today = {5, 8, 1996};

# Example - Initializing Structure Variables

## Declaration:

```c
struct location {
    int x;
    int y;
};
```

## Initialization:

```c
     struct location myrobot;
    myrobot.x = 10;
    myrobot.y = 20;
```

Or

```c
    struct location myrobot = {10,
20};
```

```c
#include <stdio.h>

struct {
    int x;
    int y;
} myrobot = {10, 20};

int main(){
    printf("x: %d\n", myrobot.x);
    printf("y: %d\n", myrobot.y);

    return 0;
}
```

# Which approach is good for declaration?

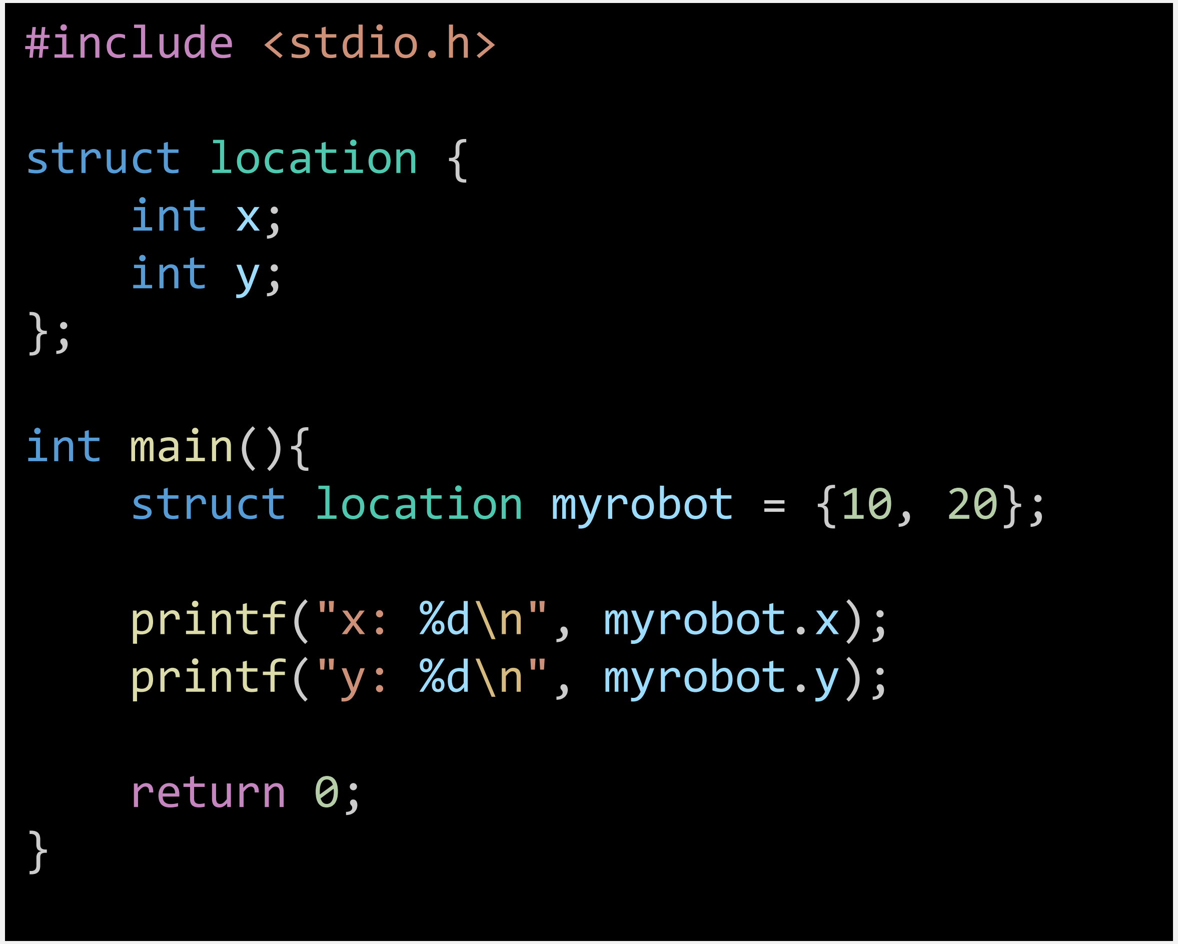
Used if number of variables are not fixed. It provides you flexibility to declare the structure variable many times.

```c
struct location{
    int x;
    int y;
};



struct location myrobot1 = {10, 20};
```

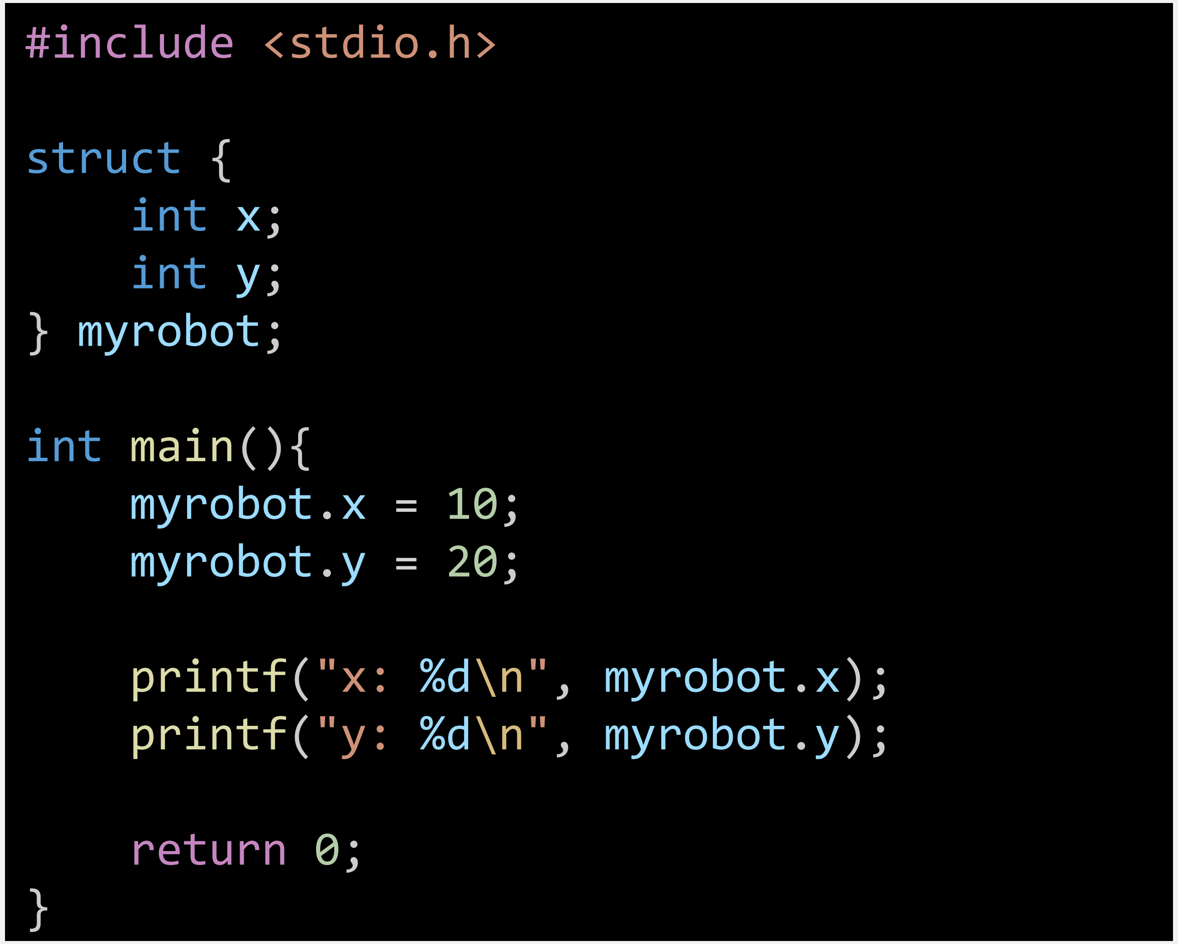
Used if number of variables are fixed. It saves your code to declare variable in main() function.

```c
struct location{
    int x;
    int y;
} myrobot1, myrobot2;
```

# Example -  Example of structure with/without tagging

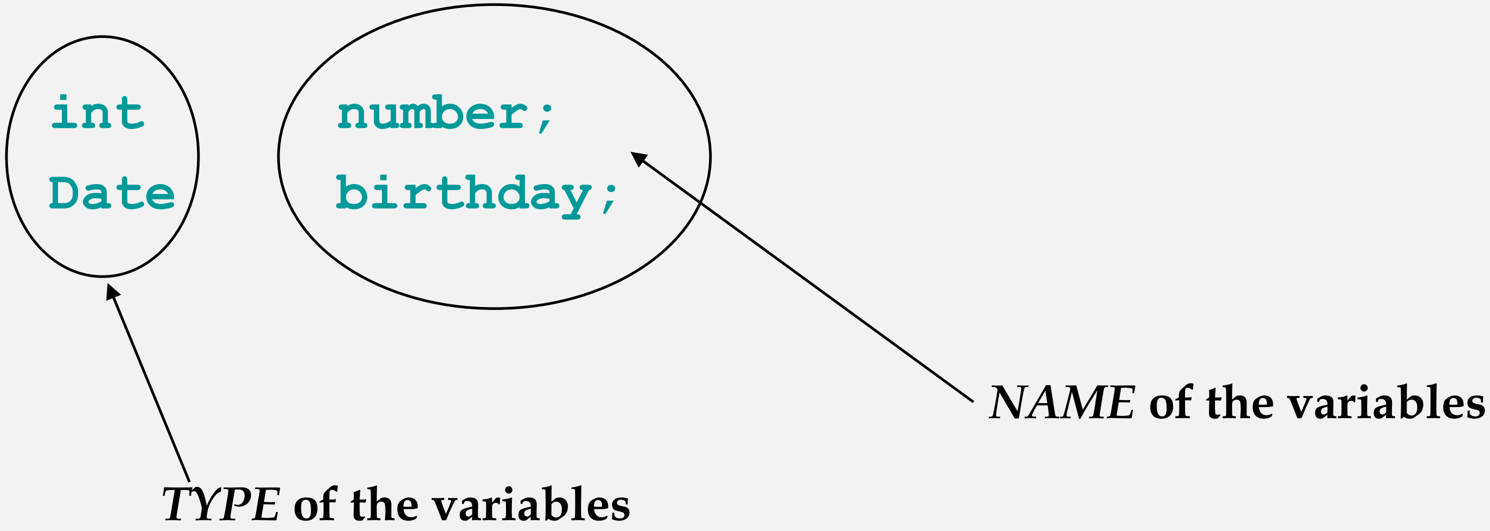


```c
#include <stdio.h>

struct location {
    int x;
    int y;
};

int main(){
    struct location myrobot = {10, 20};

    printf("x: %d\n", myrobot.x);
    printf("y: %d\n", myrobot.y);

    return 0;
}
```



```c
#include <stdio.h>

struct {
    int x;
    int y;
} myrobot;

int main(){
    myrobot.x = 10;
    myrobot.y = 20;

    printf("x: %d\n", myrobot.x);
    printf("y: %d\n", myrobot.y);

    return 0;
}
```

# Defining a Structure Variable vs Defining a "normal" Variable



```
int    number;
Date   birthday;
```

*NAME* of the variables

*TYPE* of the variables

note the consistent format :

```
<type> <variableName>;
```

# Another Example...

```
struct Date
{
   int day;
   int month;
   int year;
};

struct Date birthday  = {12, 4, 1963};
```

# Initializing Structure Type with string members

```
struct Name
{
   char first[30];
   Char last[30];
};

struct Name scientist;
Strcpy(scientist.first,” Stephen”);
Strcpy(scientist.last,” Wolfram”);
```

**Note :**
Values of the members need to be copied individually, AFTER the variable is created.

| | |
|---|---|
| "Stephen" | "Wolfram" |

# Members of Different Types

```
struct Student
{
    ... id;
    ... name;
    ... age;
    ... gender;
};
```

*The members of a* **`struct`** *need not be of the same type.*

*What should be the types of these members?*

```
student lee;
lee.id = 1234;
strcpy(lee.name,”James Lee”);
lee.age = 19;
lee.gender = 'M';
```

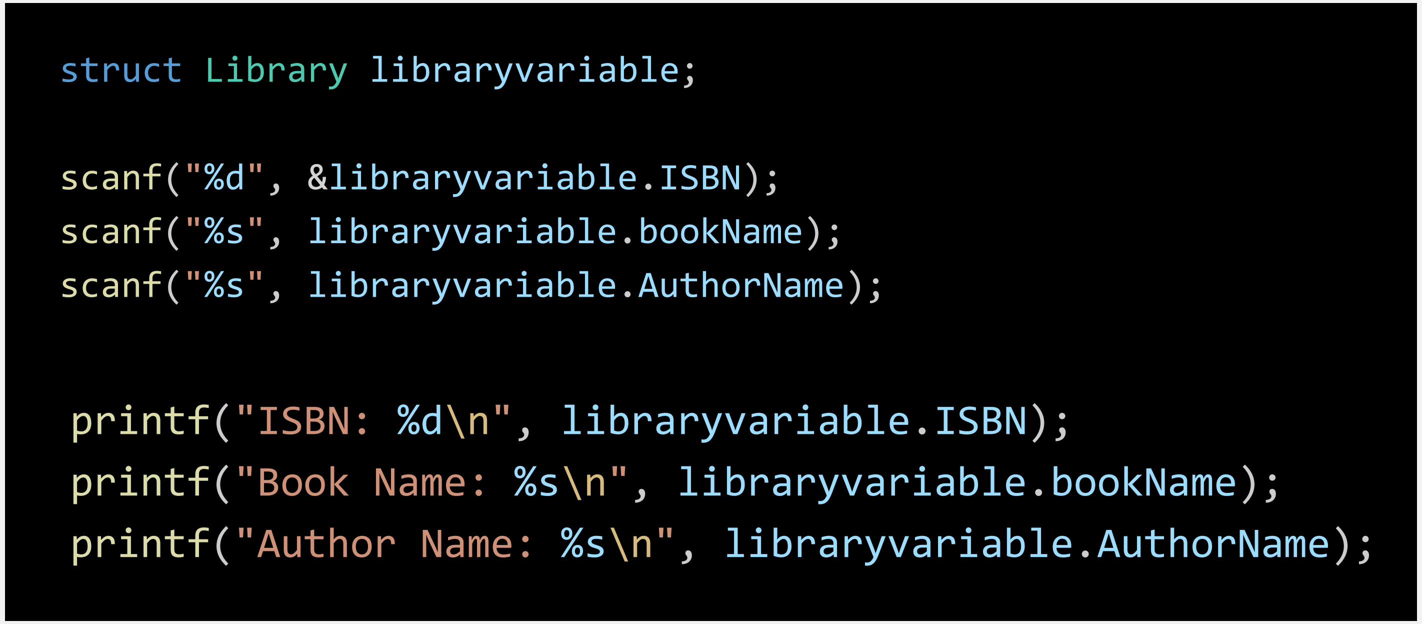
| | | | | |
|---|---|---|---|---|
| lee | 1234 | “James Lee” | 19 | ‘M’ |

# Creating structure of Library Database

| ISBN | Book Name | Author Name | Publisher | Number of Copies | Year of Publish |
|---|---|---|---|---|---|
| 978047 | Advanced Engineering Mathematics | Erwin Kreyszig | Wiley | 17 | 2011 |
| 144933 | Head First C | David Griffiths | O'Reilly | 5 | 2012 |
| 978129 | Introduction to Robotics | John Craig | Pearson | 2 | 2021 |
| 978110 | Mathematics for Machine Learning | Marc Deisenroth | Cambridge | 4 | 2020 |

```
struct Library
{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
};
```

# Accessing Structure Members



```c
struct Library libraryvariable;

scanf("%d", &libraryvariable.ISBN);
scanf("%s", libraryvariable.bookName);
scanf("%s", libraryvariable.AuthorName);


printf("ISBN: %d\n", libraryvariable.ISBN);
printf("Book Name: %s\n", libraryvariable.bookName);
printf("Author Name: %s\n", libraryvariable.AuthorName);
```

*The dot is called the “member” operator*

# Examples of Common Errors in Accessing Structures

```
struct Library
{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
} libraryvariable;
```

```
printf("Book Name: %s\n", bookName);
```

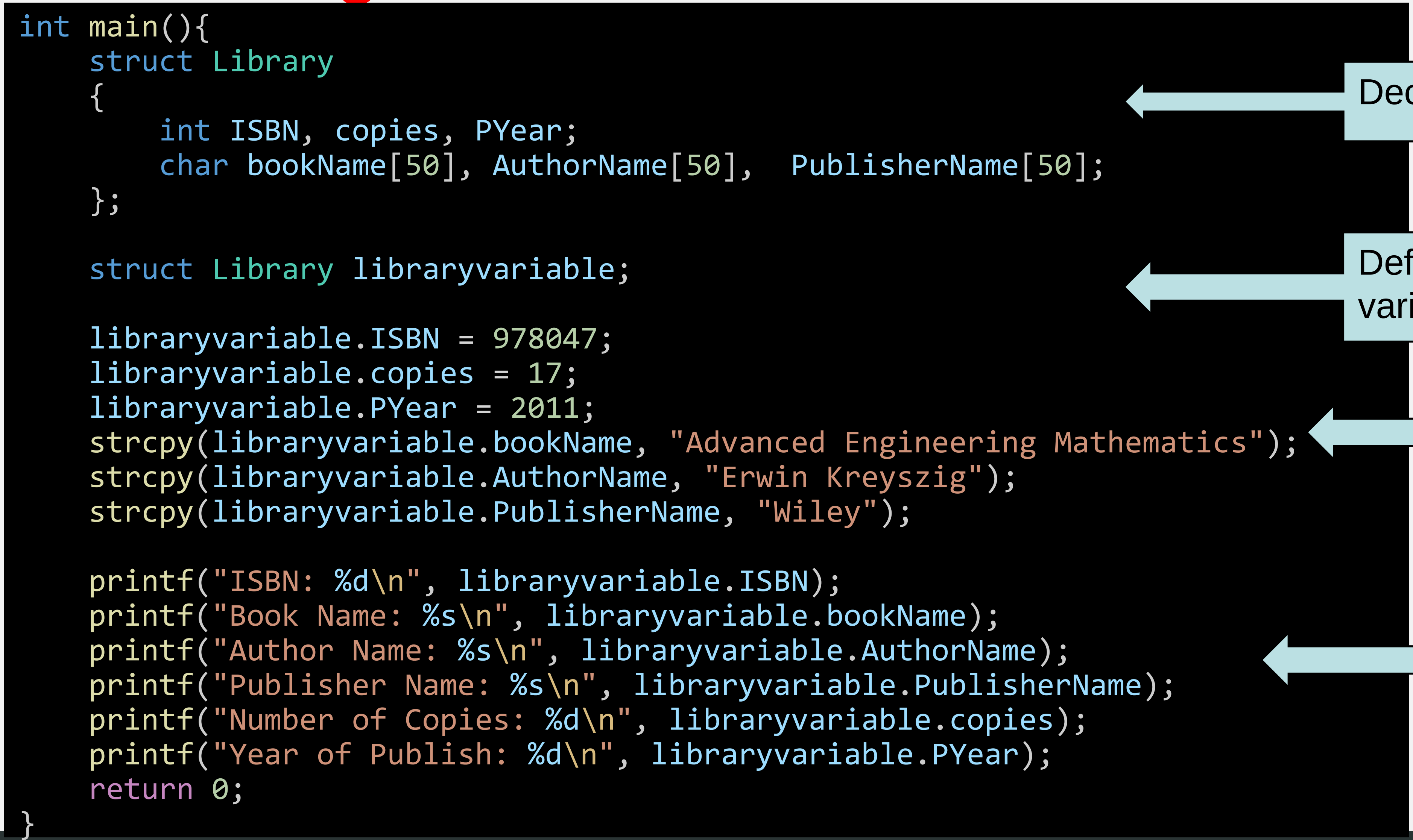
```
printf("Book Name: %s\n", Library.bookName);
```

# Common Errors in Accessing Structures

```
printf("%s\n", Libraryvariable);

//printf does not know how to handle the variable libraryVariable, as it
is not one of the built-in types. You have to give it individual bits of
libraryVariable that it can recognize and handle.
```

```
printf("ISBN: %d\n", libraryvariable.ISBN);
printf("Book Name: %s\n", libraryvariable.bookName);
//this is OK
```

# Accessing Structure Variables



```
int main(){
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[50], AuthorName[50],  PublisherName[50];
    };

    struct Library libraryvariable;

    libraryvariable.ISBN = 978047;
    libraryvariable.copies = 17;
    libraryvariable.PYear = 2011;
    strcpy(libraryvariable.bookName, "Advanced Engineering Mathematics");
    strcpy(libraryvariable.AuthorName, "Erwin Kreyszig");
    strcpy(libraryvariable.PublisherName, "Wiley");

    printf("ISBN: %d\n", libraryvariable.ISBN);
    printf("Book Name: %s\n", libraryvariable.bookName);
    printf("Author Name: %s\n", libraryvariable.AuthorName);
    printf("Publisher Name: %s\n", libraryvariable.PublisherName);
    printf("Number of Copies: %d\n", libraryvariable.copies);
    printf("Year of Publish: %d\n", libraryvariable.PYear);
    return 0;
}
```

Declaring a structure

Defining a structure variable

Initialization

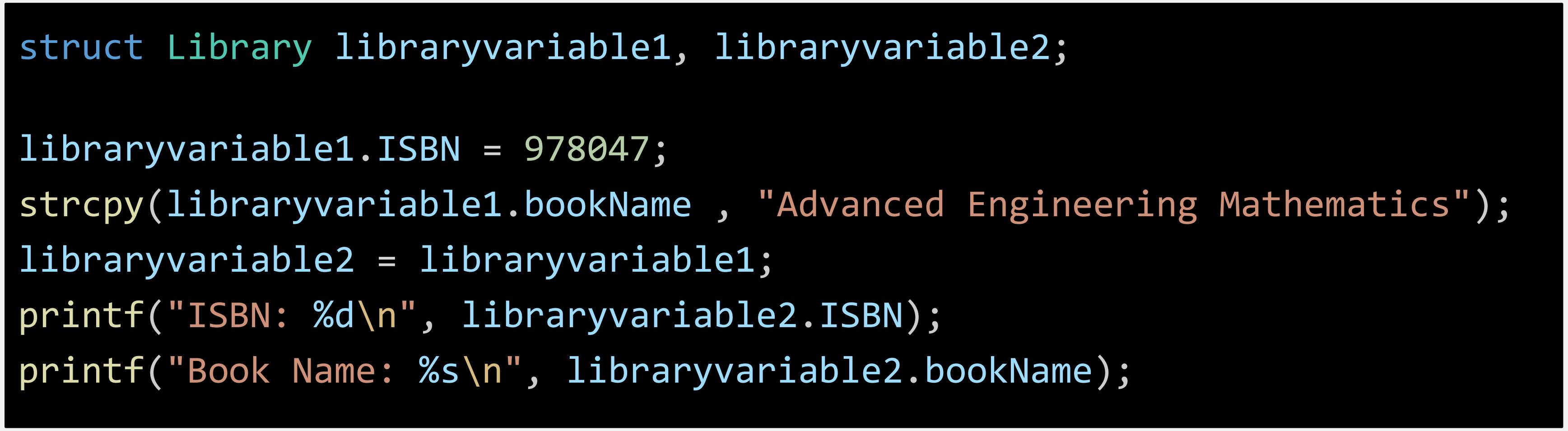
Accessing

# Accessing Structure Variables

```
int main(){
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[50], AuthorName[50],  PublisherName[50];
    };

    struct Library libraryvariable1, libraryvariable2, libraryvariable3, libraryvariable4;

    struct Library libraryvariable[4]; // alternative and easiest way


    return 0;
}
```

# Assignment to Structure Variable

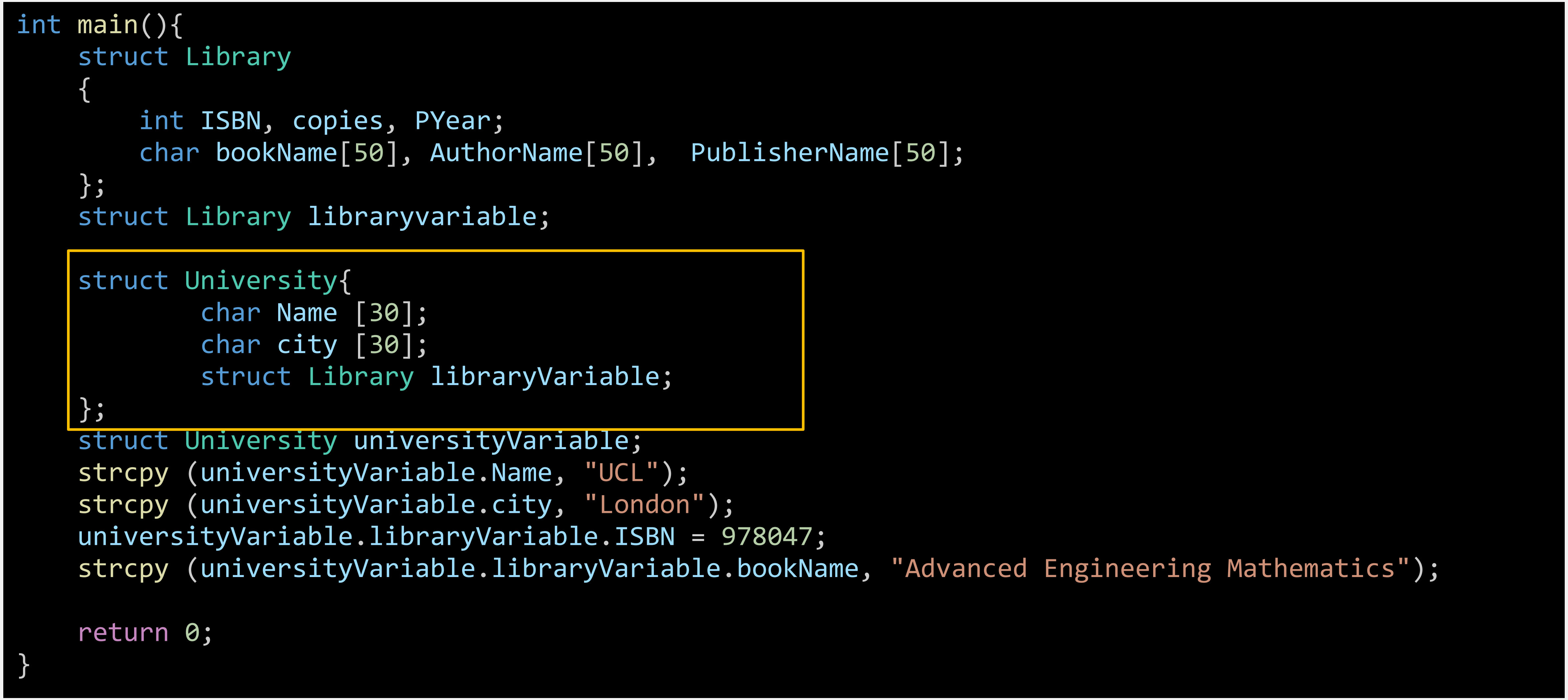
- The value of a structure variable can be assigned to another structure variable ***of the same type***, e.g:



```c
struct Library libraryvariable1, libraryvariable2;

libraryvariable1.ISBN = 978047;
strcpy(libraryvariable1.bookName , "Advanced Engineering Mathematics");
libraryvariable2 = libraryvariable1;
printf("ISBN: %d\n", libraryvariable2.ISBN);
printf("Book Name: %s\n", libraryvariable2.bookName);
```

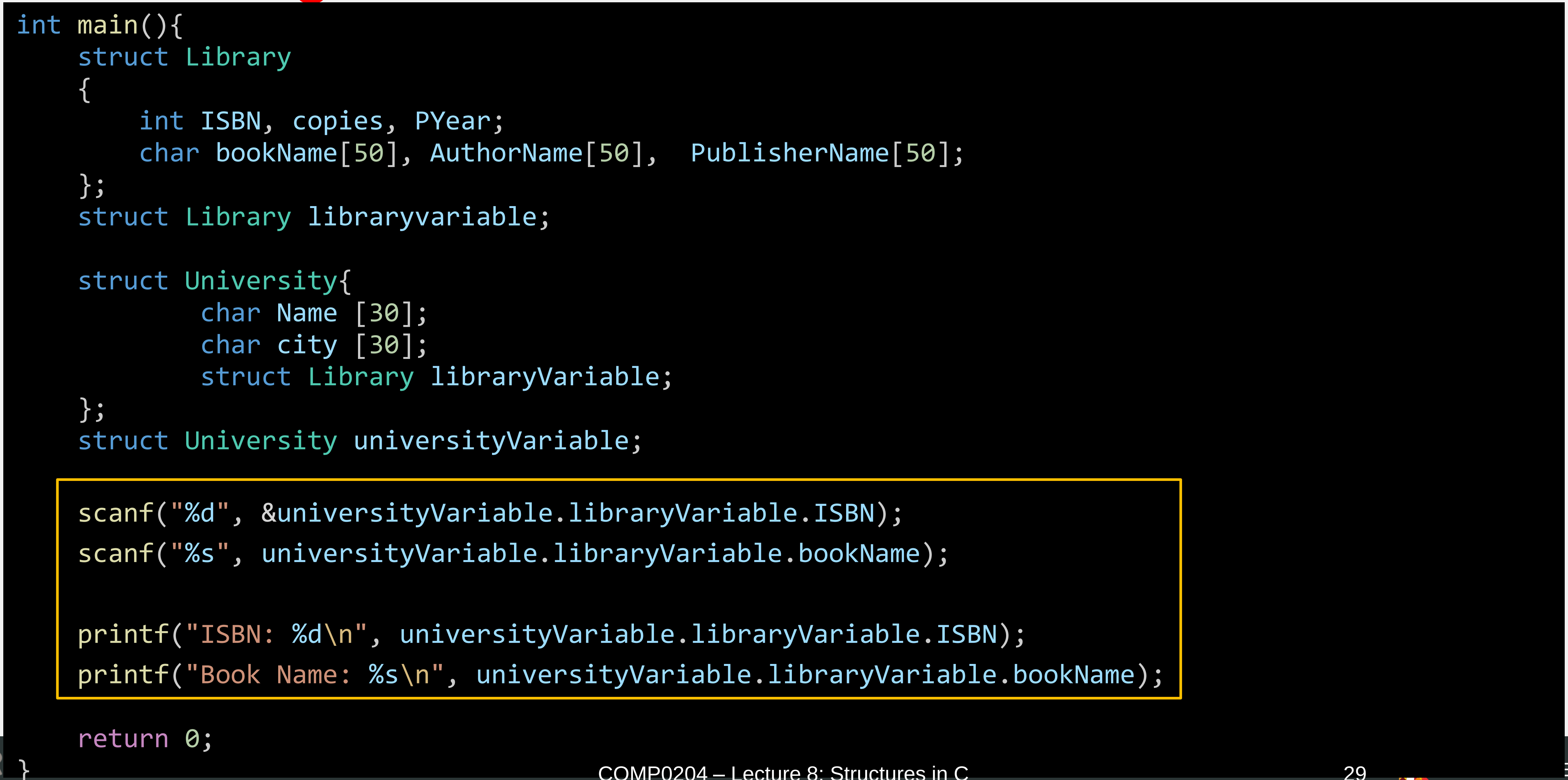
- Assignment is the only operation permitted on a structure. **We can not add, subtract, multiply or divide structures**.

# Structures within Structures



```
int main(){
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[50], AuthorName[50],  PublisherName[50];
    };
    struct Library libraryvariable;

    struct University{
            char Name [30];
            char city [30];
            struct Library libraryVariable;
    };
    struct University universityVariable;
    strcpy (universityVariable.Name, "UCL");
    strcpy (universityVariable.city, "London");
    universityVariable.libraryVariable.ISBN = 978047;
    strcpy (universityVariable.libraryVariable.bookName, "Advanced Engineering Mathematics");

    return 0;
}
```

# Accessing Structures within Structures



```
int main(){
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[50], AuthorName[50],  PublisherName[50];
    };
    struct Library libraryvariable;

    struct University{
            char Name [30];
            char city [30];
            struct Library libraryVariable;
    };
    struct University universityVariable;

    scanf("%d", &universityVariable.libraryVariable.ISBN);
    scanf("%s", universityVariable.libraryVariable.bookName);

    printf("ISBN: %d\n", universityVariable.libraryVariable.ISBN);
    printf("Book Name: %s\n", universityVariable.libraryVariable.bookName);

    return 0;
}
```

# Passing Structure to Function

It can be done in below 3 ways.

1. Passing structure to a function by value
2. Passing structure to a function by address(reference)
3. No need to pass a structure – Declare structure variable as global

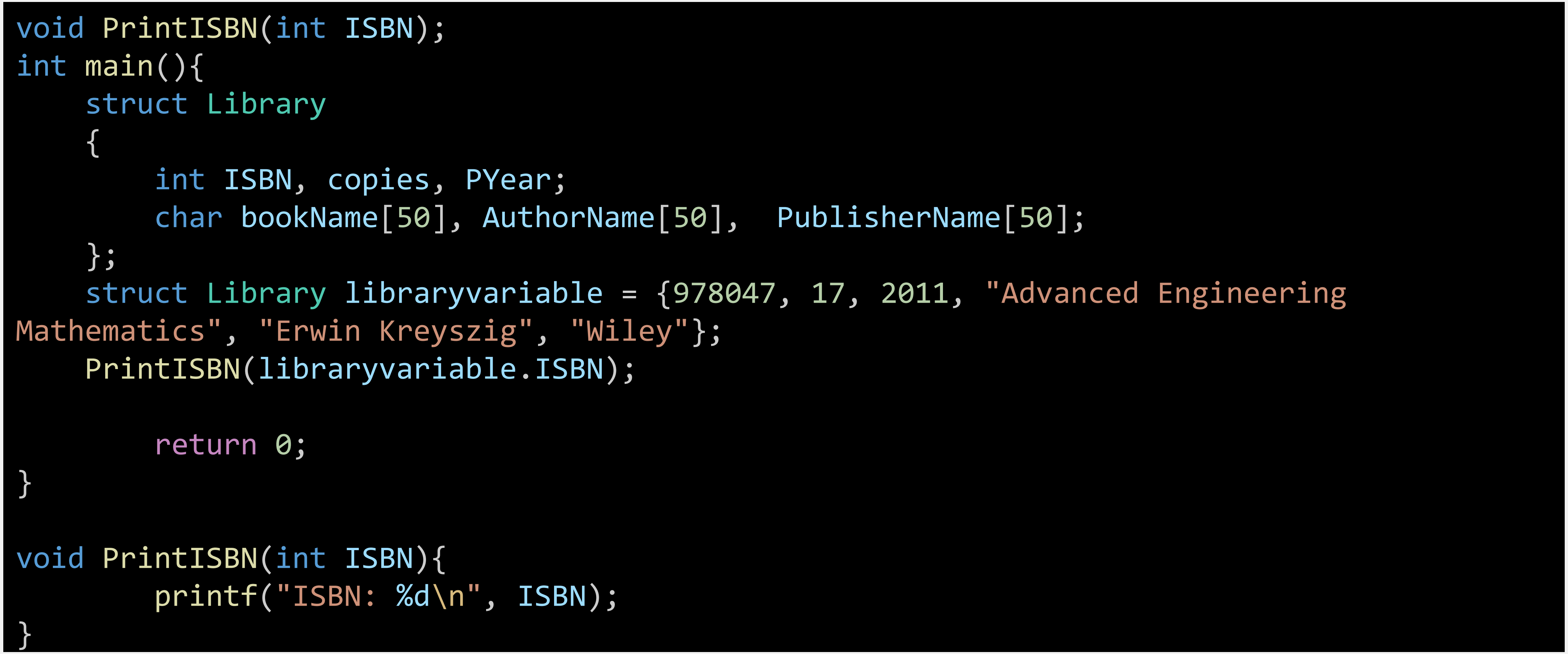
# Passing Structure Variables as Parameters

- *An individual structure member* may be passed as a *parameter to a function*, e.g. :
  - validLibraryData (libraryVariable.ISBN);

- *An entire structure variable may be passed* , e.g. :
  - validLibraryData (libraryVariable);

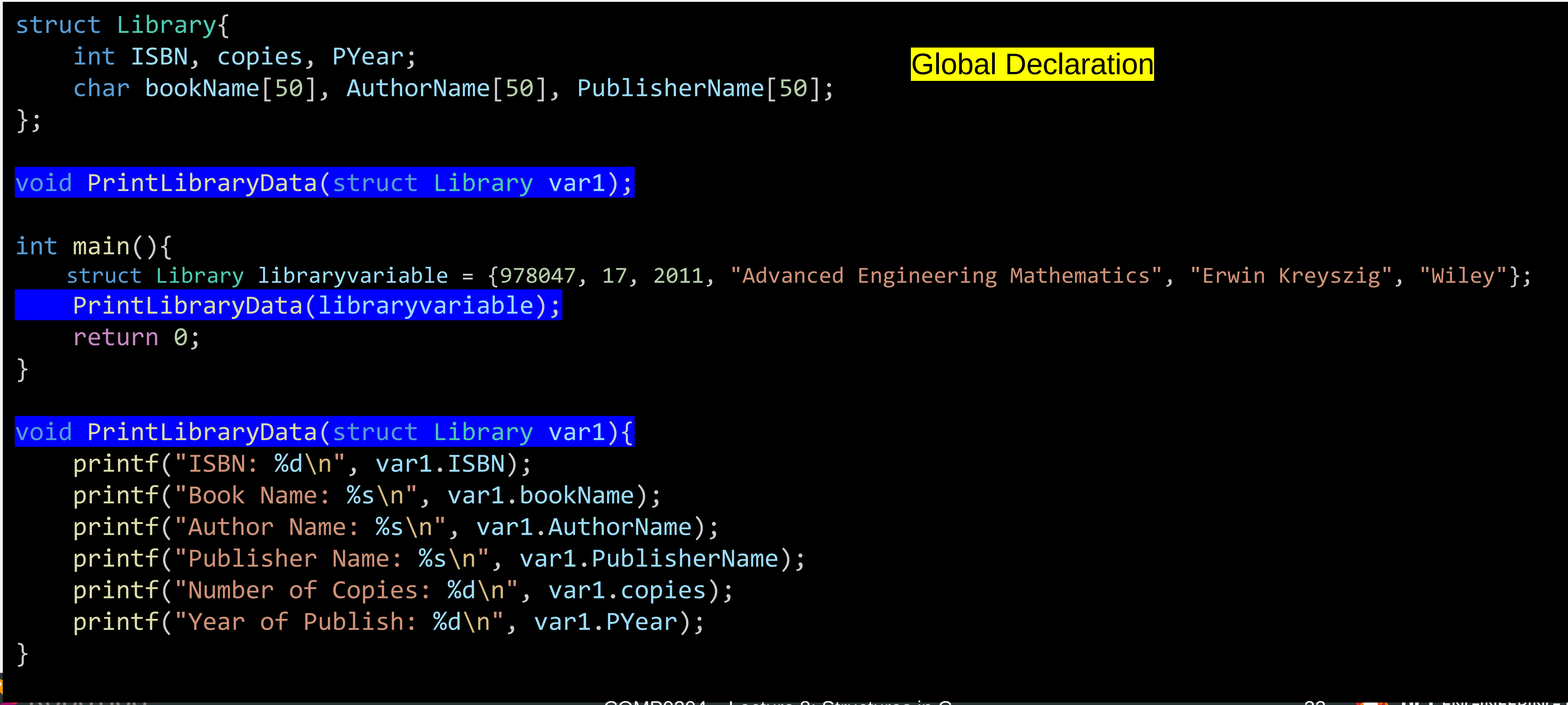
- NOTE:- Structure variable is passed by value not by reference

# Example: Passing a Structure Member



```c
void PrintISBN(int ISBN);
int main(){
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[50], AuthorName[50],  PublisherName[50];
    };
    struct Library libraryvariable = {978047, 17, 2011, "Advanced Engineering
Mathematics", "Erwin Kreyszig", "Wiley"};
    PrintISBN(libraryvariable.ISBN);

        return 0;
}

void PrintISBN(int ISBN){
        printf("ISBN: %d\n", ISBN);
}
```

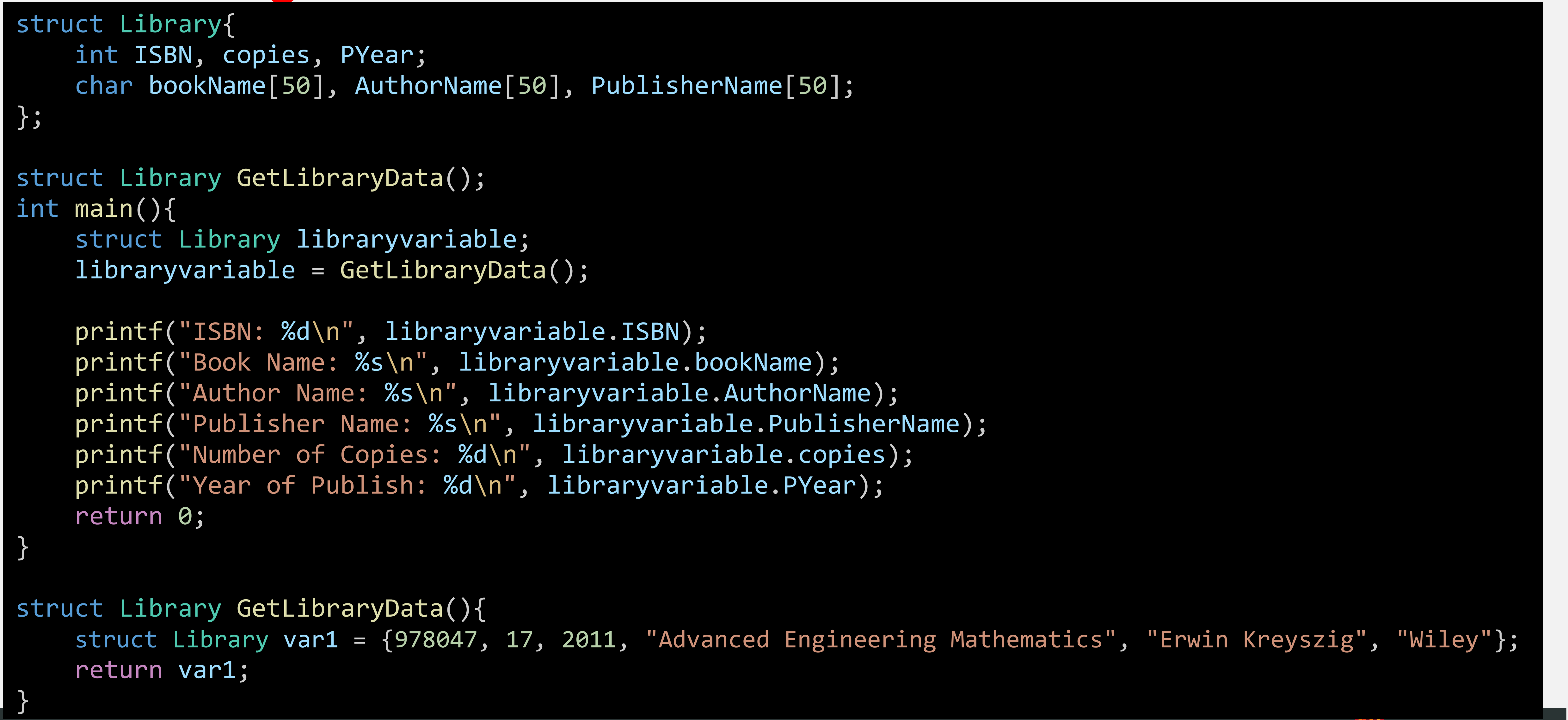
# Example: Passing an entire Structure



```c
struct Library{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
};

void PrintLibraryData(struct Library var1);

int main(){
    struct Library libraryvariable = {978047, 17, 2011, "Advanced Engineering Mathematics", "Erwin Kreyszig", "Wiley"};
    PrintLibraryData(libraryvariable);
    return 0;
}

void PrintLibraryData(struct Library var1){
    printf("ISBN: %d\n", var1.ISBN);
    printf("Book Name: %s\n", var1.bookName);
    printf("Author Name: %s\n", var1.AuthorName);
    printf("Publisher Name: %s\n", var1.PublisherName);
    printf("Number of Copies: %d\n", var1.copies);
    printf("Year of Publish: %d\n", var1.PYear);
}
```

Global Declaration

# Returning a Structure Variable



```c
struct Library{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
};

struct Library GetLibraryData();
int main(){
    struct Library libraryvariable;
    libraryvariable = GetLibraryData();

    printf("ISBN: %d\n", libraryvariable.ISBN);
    printf("Book Name: %s\n", libraryvariable.bookName);
    printf("Author Name: %s\n", libraryvariable.AuthorName);
    printf("Publisher Name: %s\n", libraryvariable.PublisherName);
    printf("Number of Copies: %d\n", libraryvariable.copies);
    printf("Year of Publish: %d\n", libraryvariable.PYear);
    return 0;
}

struct Library GetLibraryData(){
    struct Library var1 = {978047, 17, 2011, "Advanced Engineering Mathematics", "Erwin Kreyszig", "Wiley"};
    return var1;
}
```

# Pointers to structure variables

- *Pointers of structure variables can be declared like pointers to any basic data type*

```
struct Library var1, *ptrToLibrary;
ptrToLibrary = &var1;
```

- *Members of a pointer structure type variable can be accessed using (->) operator*

```
ptrToLibrary->ISBN =20;
strcpy( ptrToLibrary->bookName, “Head First C”);
```

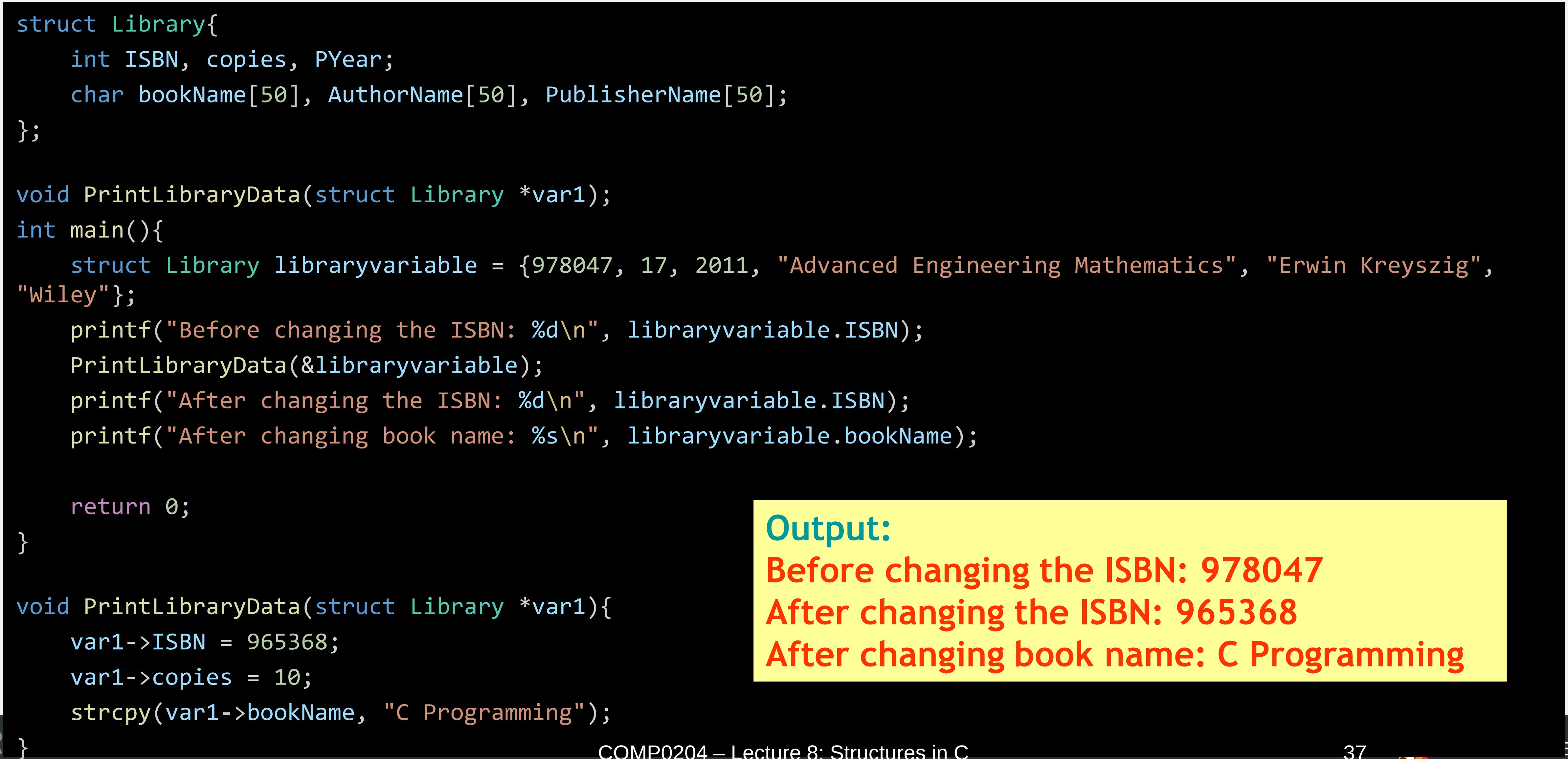
# Pointers to structure variables

```c
struct Library{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
};

int main(){
    struct Library libraryvariable = {144933, 5, 2012, "Head First C", "David
Griffiths",  "O'Reilly"};
    struct Library *PtrToLibrary;
    PtrToLibrary = &libraryvariable;

    printf("ISBN: %d\n", PtrToLibrary->ISBN);

    PtrToLibrary->ISBN = 978047;
    printf("ISBN: %d\n", PtrToLibrary->ISBN);

    return 0;
}
```

Output:

```
ISBN: 144933
ISBN: 978047
```

# Pass by Reference structure variables to Functions



```c
struct Library{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
};

void PrintLibraryData(struct Library *var1);
int main(){
    struct Library libraryvariable = {978047, 17, 2011, "Advanced Engineering Mathematics", "Erwin Kreyszig",
"Wiley"};
    printf("Before changing the ISBN: %d\n", libraryvariable.ISBN);
    PrintLibraryData(&libraryvariable);
    printf("After changing the ISBN: %d\n", libraryvariable.ISBN);
    printf("After changing book name: %s\n", libraryvariable.bookName);

    return 0;
}

void PrintLibraryData(struct Library *var1){
    var1->ISBN = 965368;
    var1->copies = 10;
    strcpy(var1->bookName, "C Programming");
}
```

**Output:**

```
Before changing the ISBN: 978047
After changing the ISBN: 965368
After changing book name: C Programming
```

# Array of Structure

- Array of structures is used to store collection of information of different data types. Each element of the array represents a **structure** variable. The array of structures is also known as collection of structures.

# Exercise: Array of Structure

- Following the library record example from this lecture, create array of structure with 3 elements.
- Enter the ISBN and book name for each record.
- Print the ISBN and book names for all three records.

# Exercise: Passing Array of Structure in Function

- Following the library record example from this lecture, create array of structure with 3 elements.
- Enter the ISBN and book name for each record.

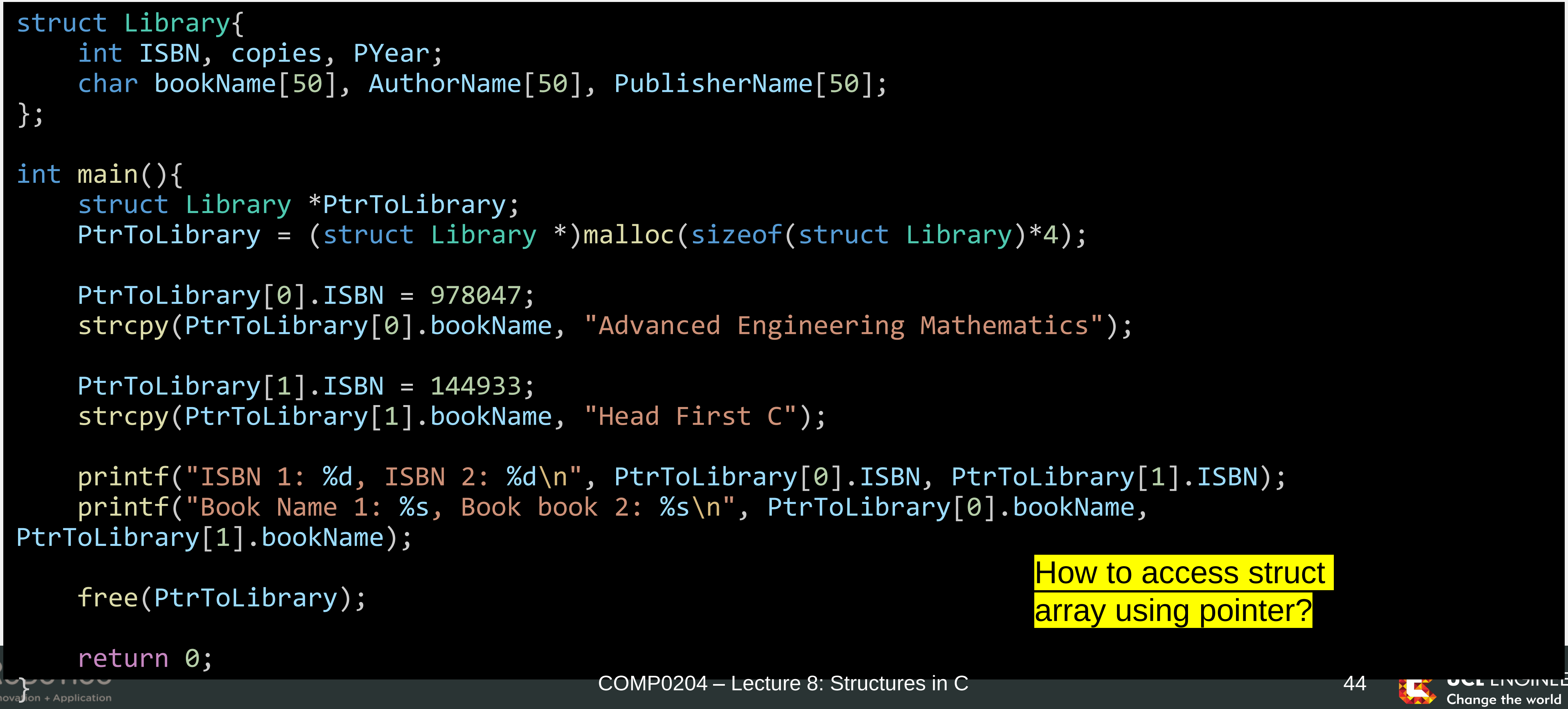
- Write a function that takes struct array as input and prints the ISBN and book names for all three records.

# Dynamic Memory Allocation (DMA) of Structure Type Variables

- ***We can also dynamic allocate the memory of any structure type variable using malloc (), realloc () functions. For example.***
  - `Library *PtrToLibrary;`
  - `PtrToLibrary = (Library *) malloc (162*10);`
- ***The above code will allocate 10 elements of Library type at execution time.***
- Very similar to
  - `float *PtrTofloat;`
  - `PtrTofloat = (float *) malloc (10*4);`

- **We can delete memory allocated at execution time using *free function***
  - `free (PtrToLibrary);`

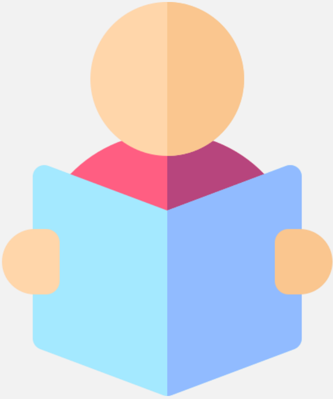
# Dynamic Memory Allocation (DMA) of Structure Type Variables



```c
struct Library{
    int ISBN, copies, PYear;
    char bookName[50], AuthorName[50], PublisherName[50];
};

int main(){
    struct Library *PtrToLibrary;
    PtrToLibrary = (struct Library *)malloc(sizeof(struct Library)*4);

    PtrToLibrary[0].ISBN = 978047;
    strcpy(PtrToLibrary[0].bookName, "Advanced Engineering Mathematics");

    PtrToLibrary[1].ISBN = 144933;
    strcpy(PtrToLibrary[1].bookName, "Head First C");

    printf("ISBN 1: %d, ISBN 2: %d\n", PtrToLibrary[0].ISBN, PtrToLibrary[1].ISBN);
    printf("Book Name 1: %s, Book book 2: %s\n", PtrToLibrary[0].bookName, 
PtrToLibrary[1].bookName);

    free(PtrToLibrary);

    return 0;
}
```

How to access struct array using pointer?

# Additional reading and coding



**Book** (E-book available in UCL Library)**:**

**C Programming for Absolute Beginner's Guide** by Greg Perry and Dean Miller: **Chapter 27**

**C in a nutshell: the definitive reference** by Peter Prinz and Tony Crawford: Chapter 10

**Tutorials:** https://codescracker.com/c/c-structures.htm

https://www.w3schools.com/c/c_structs.php