BORG AL ARAB TECHNOLOGICAL UNIVERSITY

CODE: ITS501

ADVANCED PROGRAMMING IN C

Third Year –Information Technology Program Fall 2026

LECTURE 4

INTRODUCTION TO STRUCTURES AND ENUM IN C PROGRAMMING

WHAT IS A STRUCTURE?

- **Definition**:
  A **structure** is a way to group different types of data together under one name. This helps you keep related data together in one place.
- A structure can hold many different types of data: numbers, text, or even other structures.

ADVANTAGES OF STRUCTURES

- **Organize Data**: Group related information together.
- **Data Integrity**: Keep data related to one object in one place (like a book or employee).
- **Easy to Read**: Makes the program more understandable.

BASIC STRUCTURE SYNTAX

```
struct Name {
    data_type member1;
    data_type member2;
    // other members
};
```

EXAMPLE:

- Imagine you want to store information about a **book**:
  - **Title** (text)
  - **Author** (text)
  - **Price** (number)
  - **Pages** (number)
- Instead of creating separate variables for each piece of data, you can group them together into a single structure.

STRUCTURE DECLARATION:

```
struct Book {
    char title[100];
    char author[100];
    float price;
    int pages;
};
```

struct tells C you're creating a new type of data.

Book is the name of this new data type.

The structure has 4 parts (members): title, author, price, and pages

CREATE A STRUCTURE

- You can create a structure by using the struct keyword and declare each of its members inside curly braces:

- Unlike an array, a structure can contain many different data types (int, float, char, etc.).

HOW TO ACCESS A STRUCTURE

Step 1: Define the structure

Step 2: Create a structure variable and access its members

ACCESS THE STRUCTURE,

To access the structure,

you must create a variable of it.

- Use the struct keyword inside the main() method, followed by the name of the structure and then the name of the structure variable:

```c
#include <stdio.h>
#include <string.h>

struct Book {
    char title[100];
    char author[100];
    float price;
    int pages;
};

int main() {

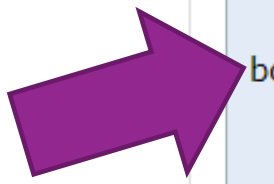
    struct Book book1;

    return 0;
}
```

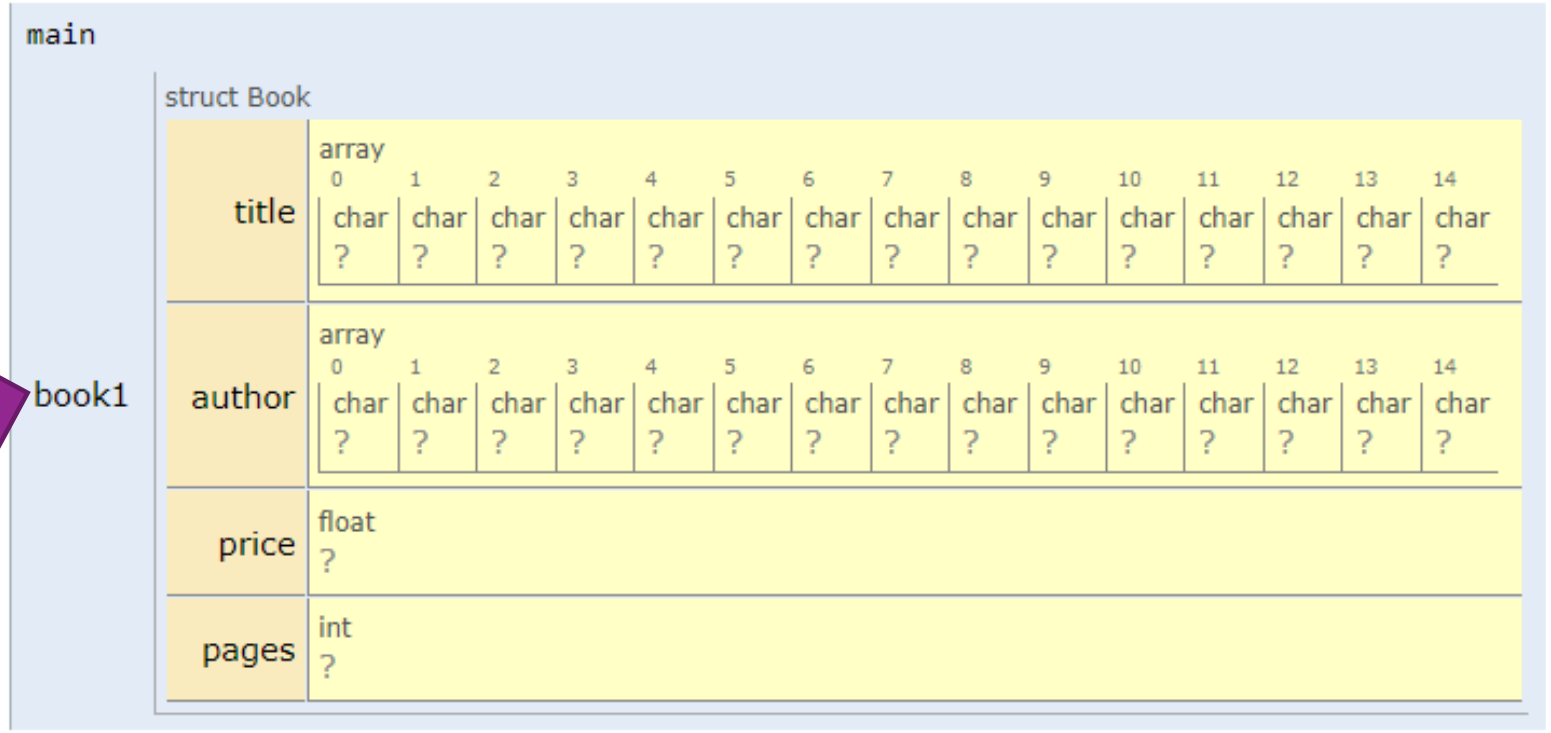
EXAMPLE

Known limitations

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct Book {
5      char title[15];
6      char author[15];
7      float price;
8      int pages;
9  };
10
11 int main() {
12
13     struct Book book1;
14
15     return 0;
16 }
```



Stack

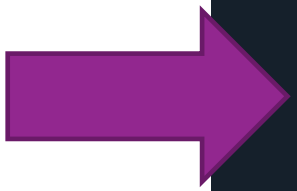


Note: ? refers to an uninitialized value

ASSIGNING VALUES TO STRUCTURE MEMBERS

Access Structure Members

- To access members of a structure, use the dot syntax (.):
- To assign values to structure members, we use the **dot operator** (.).



```c
#include <stdio.h>
#include <string.h>

struct Book {
    char title[15];
    char author[15];
    float price;
    int pages;
};

int main() {
    struct Book book1;

    book1.price = 25.99;
    book1.pages = 274;

    printf("Price: $%.2f\n", book1.price);
    printf("Pages: %d\n", book1.pages);

    return 0;
}
```

```
Price: $25.99
Pages: 274
```

WHAT ABOUT STRINGS IN STRUCTURES?

- Remember that strings in C are actually an array of characters, and unfortunately, you can't assign a value to an array like this:

```c
#include <stdio.h>
#include <string.h>

struct Book {
    char title[15];
    char author[15];
    float price;
    int pages;
};

int main() {
    struct Book book1;

    book1.title="Hello C programming";

    printf("Price: $%s\n", book1.title);

    return 0;
}
```



```
prog.c: In function ‘main’:
prog.c:14:16: error: assignment to expression with array type
   14 |     book1.title="Hello C programming";
      |                ^
```

STRINGS!

- You can use the strcpy() function and assign the value to book1.title like this:

```c
#include <stdio.h>
#include <string.h>

struct Book {
    char title[100];
    char author[100];
    float price;
    int pages;
};

int main() {
    struct Book book1;

    strcpy(book1.title, "C Programming Language");
    strcpy(book1.author, "Brian Kernighan");

    printf("Title: %s\n", book1.title);
    printf("Author: %s\n", book1.author);


    return 0;
}
```

```
Title: C Programming Language
Author: Brian Kernighan
```

PRINTING STRUCTURE MEMBERS

- **Access members** using the **dot operator** and print them with printf().

```
printf("Title: %s\n", myBook.title);
printf("Author: %s\n", myBook.author);
printf("Price: $%.2f\n", myBook.price);
printf("Pages: %d\n", myBook.pages);
```

SIMPLER SYNTAX

- You can also assign values to members of a structure variable at declaration time, in a single line.
- Just insert the values in a comma-separated list inside curly braces {}. Note that you don't have to use the strcpy() function for string values with this technique:

```c
#include <stdio.h>

struct Book {
    char title[100];
    char author[100];
    float price;
    int pages;
};

int main() {
    // Initialize structure in one line
    struct Book myBook = {"C Programming Language", "Brian Kernighan", 25.99, 274};

    // Print structure members
    printf("Title: %s\n", myBook.title);
    printf("Author: %s\n", myBook.author);
    printf("Price: $%.2f\n", myBook.price);
    printf("Pages: %d\n", myBook.pages);

    return 0;
}
```

```
Title: C Programming Language
Author: Brian Kernighan
Price: $25.99
Pages: 274
```

COPY STRUCTURES

- You can also assign one structure to another.
- In the following example, the values of s1 are copied to s2

```c
#include <stdio.h>

struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Create another structure variable
  struct myStructure s2;

  // Copy s1 values to s2
  s2 = s1;

  // Print values
  printf("%d %c %s", s2.myNum, s2.myLetter, s2.myString);

  return 0;
}
```

```
13 B Some text
```

MODIFY VALUES

- If you want to change/modify a value, you can use the dot syntax (.).
- And to modify a string value, the strcpy() function is useful again:

MODIFY VALUES

```c
#include <stdio.h>

struct Book {
    char title[100];
    char author[100];
    float price;
    int pages;
};

int main() {
    // Initialize structure in one line
    struct Book myBook = {"C Programming Language", "Brian Kernighan", 25.99, 274};

    strcpy(myBook.author,"Mohammed Ahmed");
    myBook.price=20;

    // Print structure members
    printf("Title: %s\n", myBook.title);
    printf("Author: %s\n", myBook.author);
    printf("Price: $%.2f\n", myBook.price);
    printf("Pages: %d\n", myBook.pages);

    return 0;
}
```

```
Title: C Programming Language
Author: Mohammed Ahmed
Price: $20.00
Pages: 274
```

STRUCTURE ARRAYS

- You can create an **array of structures** to store multiple books or items.

```c
struct Book {
    char title[100];
    char author[100];
    float price;
    int pages;
};

int main() {
struct Book books[3];  // Array of 3 books

// Assign values to each book
strcpy(books[0].title, "Book One");
strcpy(books[1].title, "Book Two");
strcpy(books[2].title, "Book Three");

for (int i = 0; i < 3; i++) {
    printf("Book %d: %s\n", i + 1, books[i].title);
}


    return 0;
}
```

```
Book 1: Book One
Book 2: Book Two
Book 3: Book Three
```

WHY AND WHEN TO USE ENUMS?

- Enums are used to give names to constants, which makes the code easier to read and maintain.
- Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

ENUMERATION (ENUM)

- An **enum** is a special type that represents a group of constants (unchangeable values)

```c
#include <stdio.h>

enum Level {
  LOW,
  MEDIUM,
  HIGH
};

int main() {
  // Create an enum variable and assign a value to it
  enum Level waterLevel = MEDIUM;

  // Print the enum variable
  printf("%d", waterLevel);

  return 0;
}
```

Output : 1

```
enum Level {
    LOW,
    MEDIUM,
    HIGH
};
```

- By default, the first item (LOW) has the value 0, the second (MEDIUM) has the value 1, etc.

CHANGE VALUES

```c
#include <stdio.h>

enum Level {
  LOW = 25,
  MEDIUM = 50,
  HIGH = 75
};

int main() {
  enum Level waterLevel = MEDIUM;
  printf("%d", waterLevel);

  return 0;
}
```



As you know, the first item of an enum has the value 0. The second has the value 1, and so on.

To make more sense of the values, you can easily change them:

Output: 50

- Note that if you assign a value to one specific item, the next items will update their numbers accordingly:

```c
#include <stdio.h>

enum Level {
  LOW = 5,
  MEDIUM,
  HIGH
};

int main() {
  enum Level waterLevel = MEDIUM;
  printf("%d", waterLevel);

  return 0;
}
```

Output: 6

WHAT WILL THE FOLLOWING CODE OUTPUT?

```c
enum Level {
   LOW,
   MEDIUM,
   HIGH
};

int main() {
   enum Level myVar = MEDIUM;
   printf("%d", myVar);
   return 0;
}
```

```c
#include <stdio.h>

enum Level {
   LOW = 5,
   MEDIUM,
   HIGH
};

int main() {
   enum Level myVar = HIGH;
   printf("%d", myVar);
   return 0;
}
```

- What is the default integer value assigned to the first element of an enum if no values are specified????

A)0 B)1 C)2 D)the value must be given

ANSWER

- What is the purpose of using enums in C?

A. To create floating-point constants

B. To give names to constant values, making code easier to read and maintain

C. To replace arrays

D. To create functions

DO'S

- **Use Meaningful Names for Structures and Members**
- **Initialize Structures Properly**
  - Always initialize your structures to avoid using uninitialized values.
  - You can initialize structures in one line, using the correct order of members.
- **Access Structure Members Using the Dot Operator (.)**
  - To access members of a structure, use the dot operator.
- **Use Arrays of Structures for Multiple Records**
  - If you need to store multiple records of the same type, use an array of structures.

DON'TS

- **Don't Use the Same Name for Structure and Member** :Avoid using the same name for a structure and its members. This can lead to confusion and errors.
- **Don't Forget to Initialize Structures**: Not initializing structures can result in using garbage values in your program.
- **Don't Use Structures Inside Structures Unnecessarily**: While nested structures are possible, avoid unnecessary nesting as it can make code harder to understand.

Happy Coding!