# Module 6: Creating Custom Data Types with Structs (Days 21-23)

## 1. Philosophy Focus: Knowledge Graph

You know how to work with individual pieces of data (int, double, char\*). The next logical step is to group related pieces of data into a single, logical unit. If you're modeling a user, you want to bundle their ID, name, and age together, not pass them around as three separate variables. **Structs** are C's mechanism for this, forming the foundation for building complex data models.

### **DAY 21: Defining and Using structs**

**Goal:** Learn how to define a custom data type and access its members.

1. The "Why"

Imagine you're writing a program to manage a list of books. For each book, you need to store its title, author, and year of publication.

*Without structs:*

char \*titles[] = {"The C Programming Language", "Dune"};  
char \*authors[] = {"K&R", "Frank Herbert"};  
int years[] = {1978, 1965};  
// This is messy. The data for the first book is scattered across three arrays.

With structs:

A struct (short for structure) lets you define a new type that bundles other types together.

**2. Inductive Example: Defining a Book**

#include <stdio.h>  
#include <string.h>  
  
// 1. DEFINE the new type. This is like a blueprint.  
// This doesn't create any variables yet.  
struct Book {  
 char title[100];  
 char author[50];  
 int year;  
};  
  
int main(void) {  
 // 2. DECLARE a variable of our new type.  
 // This allocates memory for one Book struct.  
 struct Book book1;   
   
 // 3. ACCESS and MODIFY its members using the dot (.) operator.  
 strcpy(book1.title, "Dune");  
 strcpy(book1.author, "Frank Herbert");  
 book1.year = 1965;  
  
 // 4. Print the members.  
 printf("Title: %s\n", book1.title);  
 printf("Author: %s\n", book1.author);  
 printf("Year: %d\n", book1.year);  
  
 return 0;  
}

**3. Discovering the Pattern**

* **Definition:** struct TypeName { members... }; This is the blueprint. It's common practice to capitalize the TypeName.
* **Declaration:** struct TypeName variable\_name; This creates an instance of the struct.
* **Access:** variable\_name.member\_name The dot (.) is used to get or set the value of a member field inside the struct.

**4. Day 21 Practice**

1. **User Profile:** Define a struct User with the following members:
   * int id;
   * char username[50];
   * int is\_active; (use 1 for true, 0 for false)
2. In main, create an instance of struct User.
3. Fill it with some data for yourself.
4. Print out the user's profile.

### **DAY 22: Structs and Arrays**

**Goal:** Learn how to create collections of your custom data types.

It's rare to only need one of something. You usually need a list. You can create an array of structs just like any other type.

**Inductive Example: A Small Library**

#include <stdio.h>  
  
struct Book {  
 char title[100];  
 char author[50];  
 int year;  
};  
  
int main(void) {  
 // Declare an array that holds 3 Book structs  
 struct Book library[3];  
  
 // Initialize the first book  
 strcpy(library[0].title, "The C Programming Language");  
 strcpy(library[0].author, "K&R");  
 library[0].year = 1978;  
  
 // Initialize the second book using a different syntax (compound literal)  
 library[1] = (struct Book){"Dune", "Frank Herbert", 1965};  
   
 // (We'll leave the third one uninitialized for now)  
  
 printf("Book 1 Title: %s\n", library[0].title);  
 printf("Book 2 Author: %s\n", library[1].author);  
  
 // Looping through an array of structs  
 printf("\n--- Full Library ---\n");  
 for (int i = 0; i < 2; i++) { // Only loop through the 2 we initialized  
 printf("%d: '%s' by %s (%d)\n", i+1, library[i].title, library[i].author, library[i].year);  
 }  
 return 0;  
}

**2. Discovering the Pattern**

* The syntax is a natural combination of what you already know:
  + Array declaration: type name[size];
  + Struct declaration: struct Book my\_book;
  + Combined: struct Book library[3];
* Accessing a member of a struct within an array combines the array index operator [] and the struct member operator .: library[0].title

**3. Day 22 Practice**

1. **To-Do List:**
   * Define a struct Task with an int id, a char description[200], and an int completed flag.
   * In main, create an array of struct Task that can hold 5 tasks.
   * Manually initialize 2-3 tasks in the array.
   * Write a for loop that iterates through the initialized tasks and prints them. For completed tasks, you could print "[X]", and for incomplete ones "[ ]".

### **DAY 23: Structs and Pointers**

**Goal:** Learn how to work with pointers to structs, which is essential for dynamic memory and efficient function calls.

When you pass a large struct to a function, the entire struct gets copied, which can be slow. It's often better to pass a *pointer* to the struct instead.

1. The Arrow Operator ->

There's a special operator for accessing members of a struct through a pointer.

**Inductive Example:**

#include <stdio.h>  
  
struct User {  
 int id;  
 char username[50];  
};  
  
// This function takes a POINTER to a User struct.  
void print\_user(struct User \*u\_ptr) {  
 printf("--- User Profile ---\n");  
   
 // Accessing members via a pointer requires the arrow operator ->  
 printf("ID: %d\n", u\_ptr->id);  
 printf("Username: %s\n", u\_ptr->username);  
  
 // The line u\_ptr->id is just syntactic sugar for (\*u\_ptr).id  
 // The arrow is much easier to read and is universally used.  
}  
  
int main(void) {  
 struct User user1 = {101, "alice"};  
   
 // Pass the ADDRESS of user1 to the function.  
 print\_user(&user1);  
  
 return 0;  
}

**2. Discovering the Pattern**

* **Declaration:** struct TypeName \*pointer\_name;
* **Get Address:** pointer\_name = &variable\_name;
* **Access via pointer:** pointer\_name->member\_name

**3. Day 23 Practice (Spaced Repetition Review)**

1. **To-Do List Function:**
   * Using the struct Task from yesterday, create a function:  
     void print\_tasks(struct Task list[], int count)  
     Note: When you pass an array to a function, it "decays" to a pointer to the first element. So struct Task list[] is equivalent to struct Task \*list.
   * This function should take the array of tasks and the number of tasks currently in the list.
   * Inside the function, loop from 0 to count and print each task's details. Use the dot . operator since you're using array indexing (list[i].description).
   * In main, create your array of tasks and call this function to print them. This is how you'll organize larger programs.