EEE: 103

Computer Programming

L11: Structures in C

Prepared by: Sk Tahmed Salim Rafid

The Problem: Storing Related Data

Let's say I want to store details of 10 students:

ID | Name | CGPA

❌ Solution 1: Multiple Variables

```
int s1id; char s1name[50]; float s1cgpa;
int s2id; char s2name[50]; float s2cgpa;
int s3id; char s3name[50]; float s3cgpa;
// ... and so on for 10 students!
```

Problems:

- ✗ Too many variables to manage
- ✗ Difficult to pass to functions
- ✗ Not scalable for large datasets

The Problem: Storing Related Data

Let's say I want to store details of 10 students:

ID | Name | CGPA

⚠️ Solution 2: Separate Arrays

```
int studentID[10];
char studentName[10][50];
float studentCGPA[10];
// Map 1-to-1: index 0 has data for student 1
```

Problems:

- ✗ Data is scattered across arrays
- ✗ Hard to keep indices synchronized
- ✗ Passing to functions requires multiple arrays

The Solution: Custom Data Structures

A struct (structure) is a user-defined data type that allows you to group different data types under a single name.

Syntax 1 - Declare struct:

```
struct struct_name {
data_type var1;
data_type var2;
};
```

📌 **Note:** To use struct, you must make an object of that struct

Making Object - Method 1:

```
struct struct_name var_name = {data1, data2};
```

// struct has 2 attributes

Making Object - Method 2:

```
struct struct_name var_name = {.var1 = data1, .var2 = data2};
```

// Designated initialization

Syntax 2 - Anonymous Struct [Not Recommended]

⚠️ **This approach has limitations and is not recommended (Not reusable)**

```
struct {
data_type var1;
data_type var2;
} struct_var1, struct_var2;
```

Explanation: struct_var1 and struct_var2 are already 2 objects

Initialize values to them:

```
struct_var1.var1 = value;
```

// Each attribute assigned individually

Syntax 3 - Using typedef [Recommended] ✓

Best practice for defining structures

```
typedef struct {
data_type var1;
data_type var2;
} own_type_name; // own_type_name can be used like int now
```

Method 1:

```
own_type_name var = {data1, data2};
```

Method 2:

```
own_type_name var = {.var1 = data1, .var2 = data2};
```

Printing:

```
printf("%specifier", var.var1);
```

Input:

```
scanf("%specifier", &var.var1);
```

Advanced Struct Operations

Creating a Pointer to Struct:

```
own_type_name* var_name = &var;
```

Accessing Data via Pointer:

```
var_name->var1
```

📌 **Note:** Needs & for scanf: scanf("%d", &var_name->var1)

Making Array of Structs:

```
own_type_name var[2] = {
{data1, data2},
{data3, data4}
```

Complete Example: Student Structure

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

typedef struct {
 int id;
 char name[50];
 float cgpa;
} Student;

int main() {
 // Create and initialize
 Student s1 = {101, "Alice", 3.85};

 // Print details
 printf("ID: %d\n", s1.id);
 printf("Name: %s\n", s1.name);
 printf("CGPA: %.2f\n", s1.cgpa);

 return 0;
}
```

Step 1 Step 2 Step 3 Step 4 Step 5

Output:

```
ID: 101
Name: Alice
CGPA: 3.85
```

Example: Using Pointer to Struct

```c
#include <stdio.h>

typedef struct {
 int id;
 char name[50];
 float cgpa;
} Student;

int main() {
 Student s1 = {101, "Alice", 3.85};
 Student *ptr = &s1; // Pointer to struct

 // Access using pointer
 printf("ID: %d\n", ptr->id);
 printf("Name: %s\n", ptr->name);
 printf("CGPA: %.2f\n", ptr->cgpa);

 return 0;
}
```

Step 1 Step 2 Step 3 Step 4 Step 5

Syntax:

```
var.member
```

vs

```
ptr->member
```

Example: Array of Structs

```c
#include <stdio.h>

typedef struct {
 int id;
 char name[50];
 float cgpa;
} Student;

int main() {
 Student students[3] = {
 {101, "Alice", 3.85},
 {102, "Bob", 3.60},
 {103, "Charlie", 3.92}
 };

 // Print all students
 for (int i = 0; i < 3; i++) {
 printf("Student %d: %s (%.2f)\n",
 students[i].id,
 students[i].name,
 students[i].cgpa);
 }

 return 0;
```

Output:

```
Student 101: Alice (3.85)
Student 102: Bob (3.60)
Student 103: Charlie (3.92)
```

Practice Problems

1

Simple Student Structure

Create a struct named student. The struct should be able to store student id, name and CGPA. Create one object of student and print all their details.

Basic

2

Pointer to Struct

Modify task-1 in such a way that created student object is stored in a pointer and pointer is used to print the details.

Basic

3

CGPA Calculator with Functions

Create a struct named student. The struct should store student id, name, number of 4 courses in an array and cgpa. CGPA will not be a user input. Create one object of this struct, take input from user for student id, name and 4 courses. Then using your custom function determine the GPA of each courses and then determine CGPA and store in the struct object.

- Assume all courses are 3 credits
- Grading: 80+ → 4.0, 60+ → 3.0, 50+ → 2.0, otherwise → 0.0
- CGPA = Average of all course GPAs

Advanced

Key Takeaways

- ✓ Structures group related data together
- ✓ Use typedef for cleaner syntax
- ✓ Access members using dot (.) operator
- ✓ Access via pointer using arrow (->) operator
- ✓ Can create arrays of structures
- ✓ Structures can be passed to functions
- ✓ Essential for organizing complex data

Thank You

Structure Your Code, Structure Your Thinking! 🎯