2. Structures (struct)

2.1 What is a Structure?

A **structure** is a user-defined data type that groups variables of different types under a single name. Think of it as creating your own custom data type.

Python vs C Comparison:

Python	C
Uses classes or dictionaries	Uses `struct`
`student = {"name": "Alice", "age": 20}`	`struct Student student;`
Dynamic typing	Static typing

2.2 Declaring a Structure

Basic Syntax:

```
struct structure_name {
    data_type member1;
    data_type member2;
    // ... more members
};
```

Example - Student Structure:

```
struct Student {
    int id;
    char name[50];
    float gpa;
    int age;
    char major[30];
};
```

Important Notes:

- Structure declaration ends with a semicolon `;`

- Members can be of any data type (including other structures)
- The structure declaration itself doesn't allocate memory

2.3 Creating Structure Variables

Method 1: After Structure Declaration

```
struct Student {
    int id;
    char name[50];
    float gpa;
};

// Create variables
struct Student student1;
struct Student student2, student3;
```

Method 2: During Structure Declaration

```
struct Student {
    int id;
    char name[50];
    float gpa;
} student1, student2;
```

Method 3: Anonymous Structure (less common)

```
struct {
    int id;
    char name[50];
    float gpa;
} student1, student2;
```

2.4 Initializing Structure Variables

Method 1: Member-by-Member Assignment

```
struct Student s1;
s1.id = 12345;
strcpy(s1.name, "Alice Johnson");  // Note: Use strcpy for strings
```

```
s1.gpa = 3.75;
```

Method 2: Initialization at Declaration

```
struct Student s1 = {12345, "Alice Johnson", 3.75};
```

Method 3: Designated Initializers (C99 and later)

```
struct Student s1 = {
    .id = 12345,
    .name = "Alice Johnson",
    .gpa = 3.75
};
```

Method 4: Partial Initialization

```
struct Student s1 = {12345};  // Only id is initialized, others are 0/NULL
```

2.5 Accessing Structure Members

Use the **dot operator (`.`)** to access structure members:

```
struct Student s1;

// Writing to members
s1.id = 12345;
s1.gpa = 3.75;
strcpy(s1.name, "Alice Johnson");

// Reading from members
printf("Student ID: %d\n", s1.id);
printf("Student Name: %s\n", s1.name);
printf("Student GPA: %.2f\n", s1.gpa);
```

2.6 Nested Structures

Structures can contain other structures as members:

```
struct Date {
    int day;
```

```
    int month;
    int year;
};

struct Student {
    int id;
    char name[50];
    float gpa;
    struct Date birthDate;  // Nested structure
};

// Usage
struct Student s1;
s1.id = 12345;
s1.birthDate.day = 15;
s1.birthDate.month = 8;
s1.birthDate.year = 2003;

printf("Birth Date: %d/%d/%d\n",
       s1.birthDate.day,
       s1.birthDate.month,
       s1.birthDate.year);
```

2.7 Array of Structures

You can create arrays of structures to handle multiple records:

```
struct Student {
    int id;
    char name[50];
    float gpa;
};

// Array of 100 students
struct Student students[100];

// Accessing elements
students[0].id = 12345;
strcpy(students[0].name, "Alice");
```

```
students[0].gpa = 3.75;

// Loop through all students
for (int i = 0; i < 100; i++) {
    printf("Student %d: %s (GPA: %.2f)\n",
           students[i].id,
           students[i].name,
           students[i].gpa);
}
```

2.8 Pointers to Structures

You can use pointers with structures:

```
struct Student s1 = {12345, "Alice", 3.75};
struct Student *ptr = &s1;

// Method 1: Using (*ptr).member
printf("ID: %d\n", (*ptr).id);

// Method 2: Using ptr->member (preferred)
printf("ID: %d\n", ptr->id);
printf("Name: %s\n", ptr->name);
printf("GPA: %.2f\n", ptr->gpa);
```

The Arrow Operator (`->`):

- `ptr->member` is equivalent to `(*ptr).member`
- Much cleaner and more readable
- Commonly used when passing structures to functions

2.9 Structures and Functions

Passing by Value:

```
void printStudent(struct Student s) {
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("GPA: %.2f\n", s.gpa);
}
```

```
// Usage
struct Student s1 = {12345, "Alice", 3.75};
printStudent(s1);  // Entire structure is copied
```

Passing by Reference (Pointer):

```
void updateGPA(struct Student *s, float newGPA) {
    s->gpa = newGPA;
}

// Usage
struct Student s1 = {12345, "Alice", 3.75};
updateGPA(&s1, 3.85);  // Pass address of structure
printf("Updated GPA: %.2f\n", s1.gpa);  // Output: 3.85
```

Returning Structures from Functions:

```
struct Student createStudent(int id, char *name, float gpa) {
    struct Student s;
    s.id = id;
    strcpy(s.name, name);
    s.gpa = gpa;
    return s;
}

// Usage
struct Student s1 = createStudent(12345, "Alice", 3.75);
```

2.10 Practical Example: Student Database

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

struct Student {
    int id;
    char name[50];
    float gpa;
    int age;
};
```

```c
// Function to input student data
void inputStudent(struct Student *s) {
    printf("Enter Student ID: ");
    scanf("%d", &s->id);

    printf("Enter Student Name: ");
    scanf(" %[^\n]", s->name);

    printf("Enter Student GPA: ");
    scanf("%f", &s->gpa);

    printf("Enter Student Age: ");
    scanf("%d", &s->age);
}

// Function to display student data
void displayStudent(struct Student s) {
    printf("\n--- Student Information ---\n");
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("GPA: %.2f\n", s.gpa);
    printf("Age: %d\n", s.age);
}

int main() {
    struct Student students[3];

    // Input data for 3 students
    for (int i = 0; i < 3; i++) {
        printf("\nEnter details for student %d:\n", i + 1);
        inputStudent(&students[i]);
    }

    // Display all students
    printf("\n\n=== All Students ===\n");
    for (int i = 0; i < 3; i++) {
        displayStudent(students[i]);
    }

    return 0;
```

```
}
```

Revision #1
Created 2025-10-05 14:43:42 UTC by DS
Updated 2025-10-05 14:44:09 UTC by DS