CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 18

Structures and Unions (2)

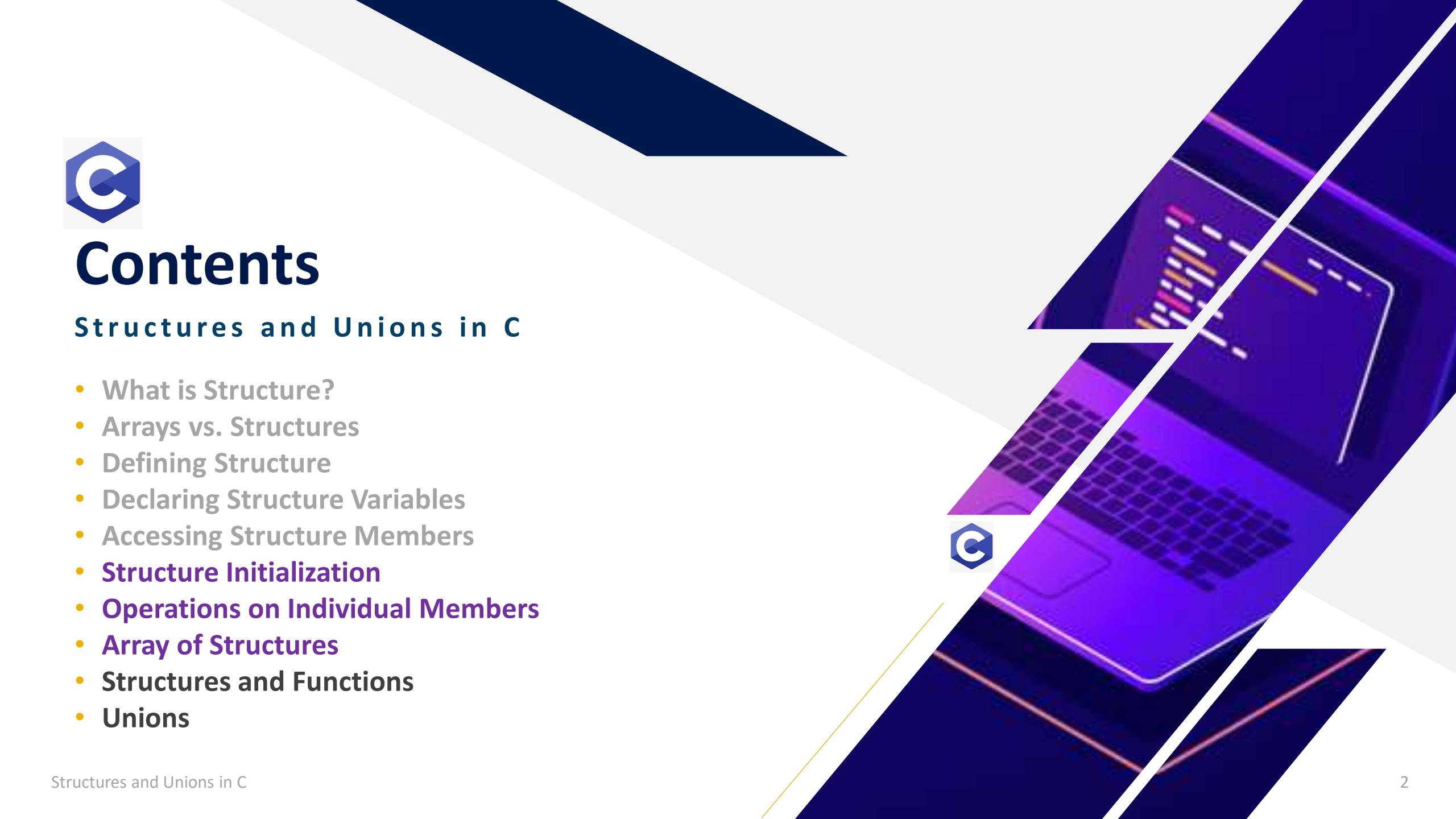
Prepared by

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering

Jatiya Kabi Kazi Nazrul Islam University, Bangladesh

www.mijanrahman.com



Contents

Structures and Unions in C

Structure Initialization

- In C, like any other data type, a structure variable can be initialized at compile time. For example:

```
int main(){
        struct {
                int weight;
                float height;
                }
        student = {60, 180.75};
        :
}
```

- This assigns the value 60 to student.weight and 180.75 to student.height. There is one-to-one correspondence between the members and their initializing values.

Structure Initialization

- A lot of variation is possible in initializing a structures.

2. The following statements initialize two structure variables. Here, it is essential to use a tag name.

```
int main(){
    struct record{
        int weight;
        float height;
        }
    struct record student1 = {60, 180.75};
    struct record student2 = {55, 170.55};
    :
}
```

Structure Initialization

3. Another method is to initialize a structure variable outside the function. For example:

```
struct record{
        int weight;
        float height;
} student1 = {60, 180.75};

int main(){
    struct record student2 = {55, 170.55};
    :
}
```

Structure Initialization

- The compile-time initialization of a structure must have the following elements:

1. The keyword **struct**.
2. The structure tag name.
3. The name of the variable to be declared.
4. The assignment operator =.
5. A set of values for the members of the structure variable, separated by commas and enclosed in braces.
6. A terminating semicolon.

Structure Initialization

- Rules for initializing structures:

1. We cannot initialize individual members inside the structure template.
2. The order of values enclosed in braces must match the order of members in the structure definition.
3. It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.
4. The uninitialized members will be assigned default values as follows:
    - Zero for integer and floating point numbers.
    - '\0' for characters and strings.

Initialization Structure Variables:

```c
#include <stdio.h>
#include <string.h>
// Define a structure called Person
struct Person {
    char name[50];
    int age;
    float height;
};

int main() {
    // Method 1: Initialize each member separately
    struct Person person1;
    strcpy(person1.name, "Rahman");
    person1.age = 30;
    person1.height = 6.0;
    // Method 2: Inline initialization
    struct Person person2 = {"Sumi", 20, 5.2};
    // Method 3: Using designated initializers
    struct Person person3 = {
        .name = "Islam",
        .age = 35,
        .height = 5.9
    };
```

Initialization Structure Variables:

```c
    // Method 4: Initializing using compound literals
    struct Person person4 = (struct Person){"Emily", 25, 5.4};

    // Print out the details of each person
    printf("Details of person1:\n");
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    printf("Height: %.2f feet\n\n", person1.height);

    printf("Details of person2:\n");
    printf("Name: %s\n", person2.name);
    printf("Age: %d\n", person2.age);
    printf("Height: %.2f feet\n\n", person2.height);

    printf("Details of person3:\n");
    printf("Name: %s\n", person3.name);
    printf("Age: %d\n", person3.age);
    printf("Height: %.2f feet\n\n", person3.height);

    printf("Details of person4:\n");
    printf("Name: %s\n", person4.name);
    printf("Age: %d\n", person4.age);
    printf("Height: %.2f feet\n\n", person4.height);
    return 0;
}
```

Terminal

```
Details of person1:
Name: Rahman
Age: 30
Height: 6.00 feet

Details of person2:
Name: Sumi
Age: 20
Height: 5.20 feet

Details of person3:
Name: Islam
Age: 35
Height: 5.90 feet

Details of person4:
Name: Emily
Age: 25
Height: 5.40 feet
```

Coping Structure Variables

1. Using Assignment Operator (=):

- We can directly assign one structure variable to another if they are of the same structure type. This performs a shallow copy of the structure.

```c
struct Person {
    char name[50];
    int age;
    float height;
};

int main() {
    struct Person person1 = {"Rahman", 30, 6.0};
    struct Person person2;

    // Copying person1 to person2
    person2 = person1;

    return 0;
}
```

Coping Structure Variables

2. Using memcpy:

- We can use the memcpy function from the <string.h> library to perform a byte-wise copy of the structure.

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

struct Person {
    char name[50];
    int age;
    float height;
};

int main() {
    struct Person person1 = {"John", 30, 6.0};
    struct Person person2;
    // Copying using memcpy
    memcpy(&person2, &person1, sizeof(struct Person));

    return 0;
}
```

Comparing Structure Variables

1. Using memcmp:

- We can use the memcmp function from the <string.h> library to compare the memory contents of two structures. This method compares structures byte by byte.

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

struct Person {
    char name[50];
    int age;
    float height;
};

int main() {
    struct Person person1 = {"John", 30, 6.0};
    struct Person person2 = {"John", 30, 6.0};
    // Comparing using memcmp
    if (memcmp(&person1, &person2, sizeof(struct Person)) == 0) {
        printf("The structures are equal.\n");
    } else {
        printf("The structures are not equal.\n");
    }

    return 0;
}
```

Comparing Structure Variables

2. Manual Comparison:

- We can compare each member of the structures individually using logical operators (==, !=, etc.).

```c
#include <stdio.h>
#include <string.h>
struct Person {
    char name[50];
    int age;
    float height;
};

int main() {
    struct Person person1 = {"John", 30, 6.0};
    struct Person person2 = {"John", 30, 6.0};

    // Manual comparison
    if (strcmp(person1.name, person2.name) == 0 &&
        person1.age == person2.age &&
        person1.height == person2.height) {
        printf("The structures are equal.\n");
    } else {
        printf("The structures are not equal.\n");
    }
    return 0;
}
```

Arrays of Structure

- In C, we can create arrays of structures to manage multiple instances of structured data. This is particularly useful when we need to work with a collection of items, each having multiple attributes.
- In such case, we can declare an array od structures, each element of the array representing a structure variable. For example:

```
struct marks{
        int sub1;
        int sub2;
        int sub3;
        }
int main(){
        struct marks student[3] = {{45, 68, 65}, {75, 55, 65}, {55, 65, 70}};
        :
}
```

Arrays of Structure

- This declares the student as an array of the threeelements student[0], student[1], and student[2], and initializes their members as follows:

```
student[0].sub1 = 45;
student[0].sub2 = 68;
student[0].sub3 = 65;
:
student[2].sub3 = 70;
```

Arrays of Structure

```c
#include <stdio.h>
#include <string.h>
struct Student {
    char name[50];
    int roll;
    float marks[3];
};
int main() {
    struct Student students[3];
    // Initialize student records
    strcpy(students[0].name, "Hossain");
    students[0].roll = 101;
    students[0].marks[0] = 85.5; // Marks for subject 1
    students[0].marks[1] = 78.0; // Marks for subject 2
    students[0].marks[2] = 92.3; // Marks for subject 3

    strcpy(students[1].name, "Sumi");
    students[1].roll = 102;
    students[1].marks[0] = 79.8;
    students[1].marks[1] = 88.5;
    students[1].marks[2] = 70.2;
```

Arrays of Structure

```
23      strcpy(students[2].name, "Prity");
24      students[2].roll = 103;
25      students[2].marks[0] = 91.2;
26      students[2].marks[1] = 83.7;
27      students[2].marks[2] = 95.0;
28
29      printf("Student Records:\n");
30      for (int i = 0; i < 3; i++) {
31          printf("\nName: %s\n", students[i].name);
32          printf("Roll Number: %d\n", students[i].roll);
33          printf("Marks for Subject 1: %.2f\n", students[i].marks[0]);
34          printf("Marks for Subject 2: %.2f\n", students[i].marks[1]);
35          printf("Marks for Subject 3: %.2f\n", students[i].marks[2]);
36      }
37      return 0;
38  }
```

Terminal

```
Student Records:
Name: Hossain
Roll Number: 101
Marks for Subject 1: 85.50
Marks for Subject 2: 78.00
Marks for Subject 3: 92.30

Name: Sumi
Roll Number: 102
Marks for Subject 1: 79.80
Marks for Subject 2: 88.50
Marks for Subject 3: 70.20

Name: Prity
Roll Number: 103
Marks for Subject 1: 91.20
Marks for Subject 2: 83.70
Marks for Subject 3: 95.00
```

Operations on Individual Members

- A member with the dot operator along with its structure variable can be treated like any other name and therefore, can be manipulated using expressions and operators. For example:

```
if (student1.roll == 103)
        student1.marks +=  10.0;
float sum = student1.marks + student2.marks;
student2.marks *=  0.5;
```

- We can also apply increment and decrement operators to numeric type members. For example:

```
student1.roll ++;
```

THE END