CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 17

Structures and Unions (1)

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

What is Structure?

- In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types.
- For example, an entity **Student** may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:
  1. Construct individual arrays for storing names, roll numbers, and marks.
  2. Use a special data structure to store the collection of different data types.

- **C provides us with an additional and simpler approach where we can use a special data structure, i.e., structure, in which, we can group all the information of different data type regarding an entity.**

What is Structure?

- In C programming, **a structure is a user-defined data type that allows us to group together different types of variables under a single name.**
- It enables us to create complex data structures to represent entities with multiple attributes.

- **Structure in C is a user-defined data type that enables us to store the collection of different data types.**
- Each element of a structure is called a member. Structures simulate the use of classes and templates as it can store various information

Arrays vs. Structures

- **Arrays and structures are both fundamental data structures in C, but they serve different purposes and have distinct characteristics.**
- Here's a comparison between arrays and structures:
- **Purpose**:
  - **Arrays**: Arrays are used to store a collection of elements of the same data type in contiguous memory locations. They are suitable for representing homogeneous data, such as a list of integers, characters, or floating-point numbers.
  - **Structures**: Structures are used to group together elements of different data types under a single name. They are suitable for representing heterogeneous data, such as a person's name, age, and address.
- **Data Type**:
  - **Arrays**: Elements of an array must be of the same data type.
  - **Structures**: Elements of a structure can be of different data types.

Arrays vs. Structures

- **Memory Allocation**:
  - **Arrays**: Elements of an array are stored in contiguous memory locations.
  - **Structures**: Members of a structure are stored in separate memory locations. The structure itself occupies memory equal to the sum of sizes of its members, possibly with some padding for alignment.
- **Accessing Elements**:
  - **Arrays**: Elements of an array are accessed using indices.
  - **Structures**: Members of a structure are accessed using dot (**.**) operator or arrow (**->**) operator (if accessing through a pointer).
- **Flexibility**:
  - **Arrays**: Arrays offer less flexibility because all elements must be of the same type and size.
  - **Structures**: Structures offer more flexibility as they can contain elements of different types and sizes.
- **Usage**:
  - **Arrays**: Useful for storing sequences of data, such as lists, matrices, or buffers.
  - **Structures**: Useful for representing complex entities or records, such as employees, students, or any object with multiple attributes.

Defining a Structure

- The **struct** keyword is used to define the structure. The syntax to define the structure in C:

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

The components:

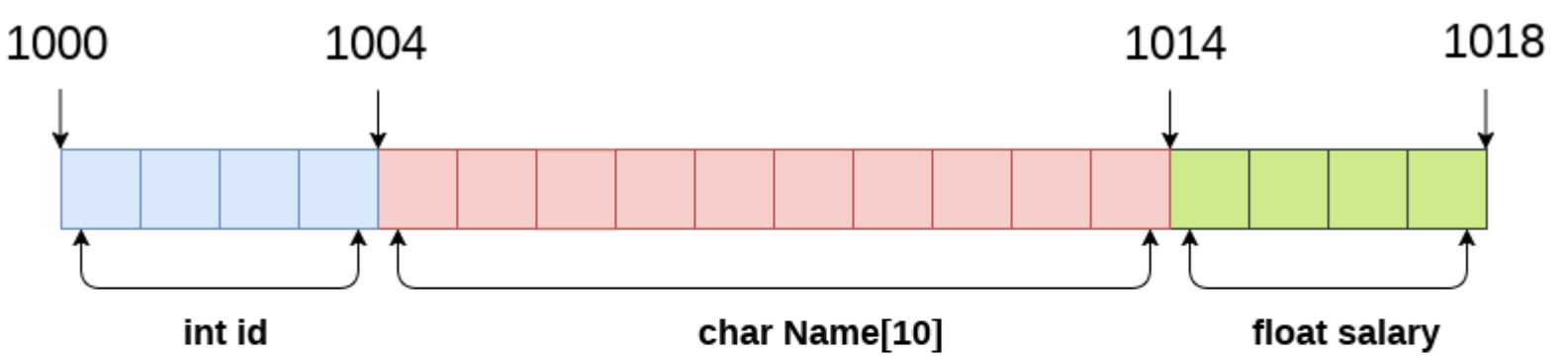
- **struct**: This is a keyword used to define a structure in C.
- **structure_name**: This is the name you give to your structure. It follows the same naming rules as variables.
- **{}**: These curly braces enclose the members of the structure.
- **data_type member1;**, **data_type member2;**, etc.: These are the members of the structure. Each member has a data type (like int, float, char, etc.) and a name.

Defining a Structure

- Let's see this example to define a structure for an entity employee in C:

```
struct employee
{   int id;
    char name[20];
    float salary;
};
```

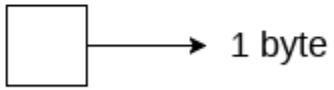
This figure shows the memory allocation of the structure employee that is defined in the above example:




```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

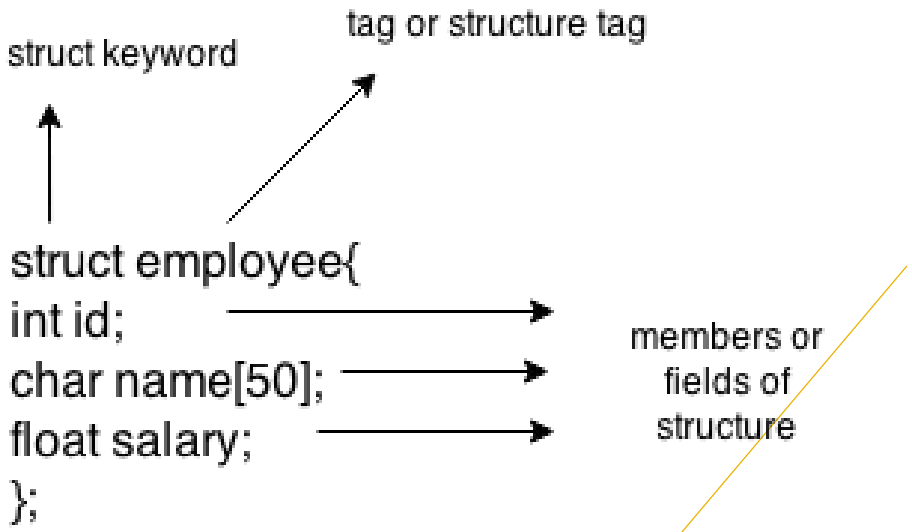
sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte



Defining a Structure

- Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



Declaring Structure Variable

- After defining a structure, we can declare variables of that type. A structure variable declaration is similar to the declaration of variable of any data types.
- It includes the following elements:
  1. The keyword **struct**
  2. The structure tag name
  3. List of variable names separated by commas
  4. A terminating semicolon

Declaring Structure Variable

- The general syntax of structure variable declaration:

  Struct structure-name variable-name;

Where:

- **struct** is the keyword used to declare a structure.
- **structure_name** is the name of the structure type.
- **variable_name** is the name you give to the structure variable.

```
// Define a structure called Person
struct Person {
        char name[50];
        int age;
        float height;
};
```

For example, a structure variable can be declared as:

```
struct Person person1;
```

Declaring Structure Variable

```
// Define a structure called Person
struct Person {
    char name[50];
    int age;
    float height;
};

For example, a structure variable can be declared as:
    struct Person person1;
```

- In this example, **struct Person** declares a structure type named **Person**. Then, within the **main** function, **struct Person person1;** declares a structure variable named **person1** of type **Person**.

Declaring Structure Variable

Ways of Declaring structure variable in C:

In C, we have several ways to declare structure variables, depending on the coding style and requirements.

Following are the common ways of declaring structure variables:

1. Using struct keyword followed by variable name: This is the most basic way to declare a structure variable.

```
struct Person person1;
```

Declaring Structure Variable

Ways of Declaring structure variable in C:

2. Using typedef with struct: This method allows us to create an alias for the structure type, making the declaration more concise.

```
typedef struct {
    char name[50];
    int age;
    float height;
} Person; Person person1;
```

3. Separate declaration and definition: We can declare the structure separately using **struct** keyword and then define the structure variable later.

```
struct Person {
    char name[50];
    int age;
    float height;
}; struct Person person1;
```

Declaring Structure Variable

Ways of Declaring structure variable in C:

4. Inline initialization: You can declare and initialize the structure variable in a single line.

```
struct Person {
    char name[50];
    int age;
    float height;
} person1 = {"John", 30, 6.0};
```

5. Declare variable at the time of defining the structure.

```
Struct Person {
    char name[50];
    int age;
    float height;
} person1, person2;
```

Declaring Structure Variable

Ways of Declaring structure variable in C:

6. Using pointers: You can declare a pointer to a structure and allocate memory dynamically using **malloc()**.

```
struct Person {
    char name[50];
    int age;
    float height;
};
struct Person *person_ptr;
person_ptr = (struct Person *)malloc(sizeof(struct Person));
```

Accessing Structure Members

- In C, we can access the members of a structure using the dot (**.**) operator or the arrow (**->**) operator if we are working with pointers to structures.

1. Using Dot Operator (for structure variables): When we have a structure variable, we can access its members using the dot (**.**) operator:

```
strcpy(person1.name, "John");
person1.age = 30;
person1.height = 6.0;
```

2.Using Arrow Operator (for pointers to structures): When we have a pointer to a structure, we can access its members using the arrow (**->**) operator:

```
strcpy(person_ptr->name, "John");
person_ptr->age = 30;
person_ptr->height = 6.0;
```

Example:

An example to illustrate how we declare a structure variable:

Using Dot Operator (for structure variables):

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

int main() {
    // Declare a structure variable of type Person
    struct Person person1;
    // Now you can access and manipulate its members
    strcpy(person1.name, "Rahman");
    person1.age = 30;
    person1.height = 6.0;

    // Print out the details
    printf("Person info:\n");
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    printf("Height: %.2f feet\n", person1.height);
    return 0;
}
```

Terminal

```
Person info:
Name: Rahman
Age: 30
Height: 6.00 feet
```

Example:

An example to illustrate how we declare a structure variable:

Using Arrow Operator (for pointers to structures):

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

int main() {
    struct Person *person_ptr;
    // Dynamically allocate memory for the structure
    person_ptr = (struct Person *)malloc(sizeof(struct Person));
    // Accessing members using arrow operator
    strcpy(person_ptr->name, "Sumi");
    person_ptr->age = 30;
    person_ptr->height = 5.2;

    printf("Person info:\n");
    printf("Name: %s\n", person_ptr->name);
    printf("Age: %d\n", person_ptr->age);
    printf("Height: %.2f feet\n", person_ptr->height);
    // Free dynamically allocated memory
    free(person_ptr);
    return 0;
}
```

Terminal

```
Person info:
Name: Sumi
Age: 30
Height: 5.20 feet
```

THE END