



جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الأمن
السيبراني
DEPARTMENT OF CYBER SECURITY

SUBJECT:

STRUCTURED PROGRAMMING

CLASS:

1ST STAGE

LECTURER:

DR. ABDULKADHEM A. ABDULKADHEM

LECTURE: (10)

Structures (struct) in C++



Lecture Objectives

By the end of this lecture, students should be able to:

- Understand the syntax and use of structures (`struct`) in C++
- Define and access members of a structure
- Initialize structure variables
- Use arrays of structures and nested structures
- Compare structures to classes (briefly)
- Understand memory layout implications

1. Introduction to `struct`

A `struct` (short for *structure*) in C++ is a user-defined data type that groups variables of **different types** into a single unit.

□ Why use `struct`?

- To represent **real-world entities** (e.g., Student, Book, Point)
- To create **records** containing multiple data fields
- Useful in organizing and handling **composite data**

2. Basic Syntax

```
struct StructName {  
    data_type member1;  
    data_type member2;  
    // ...  
};
```

Example:

```
#include <iostream>  
using namespace std;  
  
struct Student {  
    int id;  
    string name;  
    float gpa;  
};  
  
int main() {  
    Student s1;  
    s1.id = 1001;  
    s1.name = "Ali";  
    s1.gpa = 3.75;  
}
```



```
cout << "ID: " << s1.id << endl;  
cout << "Name: " << s1.name << endl;  
cout << "GPA: " << s1.gpa << endl;  
  
return 0;  
}
```

3. Declaring and Initializing Structures

- ☐ Variable declaration after struct:

```
struct Point {  
    int x, y;  
};  
  
Point p1 = {10, 20}; // aggregate initialization
```

- ☐ With constructor-style initializer (C++11 and above):

```
Point p2{30, 40};
```

4. Accessing Members

Use the **dot operator** . to access members:

```
cout << p1.x; // outputs 10
```

5. Array of Structures

```
Student classList[3];  
  
classList[0] = {1001, "Ali", 3.7};  
classList[1] = {1002, "Zainab", 3.9};
```

6. Passing Structures to Functions

By value:

```
void display(const Student s) {  
    cout << s.name;  
}
```



By reference (preferred):

```
void display(const Student &s) {  
    cout << s.name;  
}
```

7. Nested Structures

```
struct Date {  
    int day, month, year;  
};  
  
struct Employee {  
    int id;  
    string name;  
    Date dob; // structure inside structure  
};
```

8. struct VS class

Feature	struct	class
Default access	public	private
Inheritance	Supported	Supported
Use case	Data grouping	Full OOP

□ In modern C++, `struct` and `class` are **nearly identical** except for default access modifier.

9. Memory Layout of Structs

Each member is laid out **sequentially** in memory. Compiler may add **padding** for alignment.

Example:

```
struct Mixed {  
    char c;    // 1 byte  
    int x;     // 4 bytes  
};
```

Memory = likely 8 bytes due to padding (1 + 3 + 4)



10. Practical Example: Structure and Function

```
#include <iostream>
using namespace std;

struct Book {
    int id;
    string title;
    float price;
};

void printBook(Book b) {
    cout << b.id << " - " << b.title << " - $" << b.price << endl;
}

int main() {
    Book b1 = {101, "C++ Programming", 19.99};
    printBook(b1);
}
```

Summary

- `struct` groups different types into one unit
- Use dot (.) to access members
- Pass by reference to functions for efficiency
- Useful for modeling real-world objects
- `struct` and `class` only differ in default member access

Exercises

1. Define a structure `Car` with brand, model, and year.
2. Create an array of 3 students and print their info.
3. Define a `Rectangle` struct and a function to calculate area