| | | | |
|---|---|---|---|
| **Course Name:** | **Programming in C** | **Semester:** | **II** |
| **Date of Performance:** | **10-03-2025** | **DIV/ Batch No:** | **C-5 (3)** |
| **Student Name:** | **MAYURI MANOJ KUMBHAR** | **Roll No:** | [redacted] |

# Experiment No: 7
# Title: Structures and unions

**Aim and Objective of the Experiment:**

Write a program in C to demonstrate use of structures and unions.

**COs to be achieved:**

**CO4: Design modular programs using functions and the use of structure and union.**

**Theory:**

## Introduction to Structures

A structure is a user-defined data type in C that groups variables of different types under a single name. Structures are used when you need to store multiple related pieces of data, such as information about a student, employee, or product, where each field might have a different data type (e.g., integers, floats, and characters).
Example: A structure could be defined to store a student's name, age, and grade.

### Declaring and Defining a Structure

To declare and define a structure in C, you first use the struct keyword, followed by a structure name, and the members enclosed within curly braces {}. Each member can be of a different data type.
Syntax:

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

Example:

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

This defines a structure Student with three members: a string for the name, an integer for the age,

and a float for the grade.

**Structure Initialization**

Structures can be initialized at the time of declaration or later by assigning values to their members individually. If initialization during declaration, values for the members are assigned in the same order as their declaration.

Syntax for initialization:

```
struct structure_name variable_name = {value1, value2, ...};
```

Example:

```
struct Student student1 = {"John", 20, 85.5};
```

Alternatively, individual members can be initialized after declaration:

```
student1.age = 21;
strcpy(student1.name, "Alice");
student1.grade = 90.0;
```

**Accessing and Displaying Structure Members**

Structure members can be accessed using the dot (.) operator. The member values can be printed or manipulated as required.

Syntax:

```
variable_name.member_name
```

Example:

```
printf("Name: %s\n", student1.name);
printf("Age: %d\n", student1.age);
printf("Grade: %.2f\n", student1.grade);
```

If a structure is pointed to by a pointer, the arrow (->) operator is used to access members.

Example:

```
struct Student *ptr = &student1;
printf("Name: %s\n", ptr->name);
```

**Array of Structures**

An array of structures is used when you want to store multiple instances of a structure. Each element of the array is a structure.

Syntax: `struct structure_name array_name[size];`

Example:

```
struct Student students[3];
students[0].age = 20;
strcpy(students[0].name, "Alice");
```

```
students[0].grade = 90.0;
```
To loop through an array of structures, you can use a for loop:
```
for (int i = 0; i < 3; i++) {
   printf("Name: %s, Age: %d, Grade: %.2f\n", students[i].name, students[i].age, students[i].grade);
}
```

**Introduction to Unions**

A union is a user-defined data type similar to a structure, but with one key difference: all members of a union share the same memory location. This means that at any given time, only one member of the union can hold a value, making it more memory efficient when you don't need to store multiple values simultaneously.

Syntax:
```
union union_name {
   data_type member1;
   data_type member2;
   // more members
};
```

Example:
```
union Data {
   int i;
   float f;
   char str[20];
};
```
In the above example, the Data union can store an integer, a float, or a string, but only one of these at a time. The memory allocated for all the members of the union is the size of the largest member.

**Accessing Members of a Union**

Just like structures, union members are accessed using the dot (.) operator. However, because all members share the same memory space, modifying one member will overwrite the other members' values.

Example:
```
union Data data;
data.i = 10;    // Valid
data.f = 3.14;  // Overwrites 'i'
data.str = "Hello";  // Overwrites 'f'
```

**Problem Statements:**

Design a C program to manage employee data using structures and unions. The program should allow the following functionalities:

1. **Employee Data Input:**
   - Each employee should have the following common attributes:
     - Employee ID (integer)
     - Name (string)
     - Age (integer)
     - Department (string)
     - Basic Salary (float)
   - Depending on the employee's role, additional attributes should be stored:
     - For **Sales Employees**:
       - Commission (float)
       - Sales Target (float)
     - For **Technical Employees**:
       - Project Name (string)
       - Project Allowance (float)
   - Use a **union** to store role-specific data efficiently.
2. **Employee Data Display:**
   - Display all employee details, including role-specific information, in a formatted manner.
3. **Calculate Total Salary:**
   - For each employee, calculate the total salary based on their role:
     - For **Sales Employees**: Total Salary = Basic Salary + Commission
     - For **Technical Employees**: Total Salary = Basic Salary + Project Allowance
4. **Search Employee by ID:**
   - Allow the user to search for an employee by their Employee ID and display their details.
5. **Update Employee Data:**
   - Allow the user to update specific details of an employee (e.g., name, age, department, or role-specific data).
6. **Delete Employee Data:**
   - Allow the user to delete an employee's record by their Employee ID.

**Requirements:**

1. Use a **structure** to represent an employee with common attributes.
2. Use a **union** to store role-specific attributes (either for sales or technical employees).
3. Use an **enum** to differentiate between employee roles (e.g., SALES, TECHNICAL).
4. Implement dynamic memory allocation to store employee records.
5. Provide a menu-driven interface for the user to perform the above operations.

**Code :**

```
#include <stdio.h>
typedef enum {
```

```
    SALES,
    TECHNICAL
} Role;

typedef union {
    struct {
        float commission;
        float salesTarget;
    } sales;

    struct {
        char projectName[50];
        float projectAllowance;
    } technical;
} RoleData;

typedef struct {
    int employeeID;
    char name[50];
    int age;
    char department[30];
    float basicSalary;
    Role role;
    RoleData roleData;
} Employee;

void inputEmployeeData(Employee *emp) {
    printf("Enter Employee ID: ");
    scanf("%d", &emp->employeeID);

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

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

    printf("Enter Department: ");
    scanf(" %[^\n]", emp->department);

    printf("Enter Basic Salary: ");
    scanf("%f", &emp->basicSalary);

    printf("Enter Role (0 for SALES, 1 for TECHNICAL): ");
    scanf("%d", (int *)&emp->role);
```

```
    if (emp->role == SALES) {
        printf("Enter Commission: ");
        scanf("%f", &emp->roleData.sales.commission);

        printf("Enter Sales Target: ");
        scanf("%f", &emp->roleData.sales.salesTarget);
    } else if (emp->role == TECHNICAL) {
        printf("Enter Project Name: ");
        scanf(" %[^\n]", emp->roleData.technical.projectName);

        printf("Enter Project Allowance: ");
        scanf("%f", &emp->roleData.technical.projectAllowance);
    }
}

// Function to Calculate Total Salary
float calculateTotalSalary(const Employee *emp) {
    if (emp->role == SALES) {
        return emp->basicSalary + emp->roleData.sales.commission;
    } else if (emp->role == TECHNICAL) {
        return emp->basicSalary + emp->roleData.technical.projectAllowance;
    }
    return emp->basicSalary;
}


void displayEmployeeData(const Employee *emp) {
    printf("\nEmployee ID: %d\n", emp->employeeID);
    printf("Name: %s\n", emp->name);
    printf("Age: %d\n", emp->age);
    printf("Department: %s\n", emp->department);
    printf("Basic Salary: %.2f\n", emp->basicSalary);

    if (emp->role == SALES) {
        printf("Role: Sales\n");
        printf("Commission: %.2f\n", emp->roleData.sales.commission);
        printf("Sales Target: %.2f\n", emp->roleData.sales.salesTarget);
    } else if (emp->role == TECHNICAL) {
        printf("Role: Technical\n");
        printf("Project Name: %s\n", emp->roleData.technical.projectName);
        printf("Project Allowance: %.2f\n", emp->roleData.technical.projectAllowance);
    }

    printf("Total Salary: %.2f\n", calculateTotalSalary(emp));
}
```

```
Employee* searchEmployeeByID(Employee *employees, int count, int id) {
    for (int i = 0; i < count; i++) {
        if (employees[i].employeeID == id) {
            return &employees[i];
        }
    }
    return NULL;
}


void updateEmployeeData(Employee *emp) {
    printf("Updating data for Employee ID: %d\n", emp->employeeID);

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

    printf("Enter New Age: ");
    scanf("%d", &emp->age);

    printf("Enter New Department: ");
    scanf(" %[^\n]", emp->department);

    printf("Enter New Basic Salary: ");
    scanf("%f", &emp->basicSalary);

    if (emp->role == SALES) {
        printf("Enter New Commission: ");
        scanf("%f", &emp->roleData.sales.commission);

        printf("Enter New Sales Target: ");
        scanf("%f", &emp->roleData.sales.salesTarget);
    } else if (emp->role == TECHNICAL) {
        printf("Enter New Project Name: ");
        scanf(" %[^\n]", emp->roleData.technical.projectName);

        printf("Enter New Project Allowance: ");
        scanf("%f", &emp->roleData.technical.projectAllowance);
    }
}


void deleteEmployee(Employee employees[], int *count, int id) {
    for (int i = 0; i < *count; i++) {
```

```
        if (employees[i].employeeID == id) {
            for (int j = i; j < *count - 1; j++) {
                employees[j] = employees[j + 1];
            }
            (*count)--;
            printf("Employee with ID %d deleted successfully.\n", id);
            return;
        }
    }
    printf("Employee with ID %d not found!\n", id);
}

int main() {
    Employee employees[10]; // Array to hold up to 10 employees
    int count = 0;          // Number of employees
    int choice, id;

    while (1) {
        // Display Menu
        printf ("\n--- Employee Management System ---\n");
        printf ("1. Add Employee\n");
        printf ("2. Display All Employees\n");
        printf ("3. Display Total Salaries of All Employees\n");
        printf ("4. Search Employee by ID\n");
        printf ("5. Update Employee Data\n");
        printf ("6. Delete Employee Data\n");
        printf ("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) { // Add Employee
            if (count < 10) {
                inputEmployeeData(&employees[count]);
                count++;
            } else {
                printf("Employee list is full!\n");
            }
        } else if (choice == 2) { // Display All Employees
            for (int i = 0; i < count; i++) {
                displayEmployeeData(&employees[i]);
            }
        } else if (choice == 3) { // Display Total Salaries of All Employees
            for (int i = 0; i < count; i++) {
                printf("Employee ID: %d, Total Salary: %.2f\n", employees[i].employeeID,
calculateTotalSalary(&employees[i]));
```

```
            }
        } else if (choice == 4) { // Search Employee by ID
            printf("Enter Employee ID to search: ");
            scanf("%d", &id);
            Employee *emp = searchEmployeeByID(employees, count, id);
            if (emp) {
                displayEmployeeData(emp);
            } else {
                printf("Employee not found!\n");
            }
        } else if (choice == 5) { // Update Employee Data
            printf("Enter Employee ID to update: ");
            scanf("%d", &id);
            Employee *emp = searchEmployeeByID(employees, count, id);
            if (emp) {
                updateEmployeeData(emp);
            } else {
                printf("Employee not found!\n");
            }
        } else if (choice == 6) { // Delete Employee Data
            printf("Enter Employee ID to delete: ");
            scanf("%d", &id);
            deleteEmployee(employees, &count, id);
        } else if (choice == 7) { // Exit
            printf("Exiting program. Goodbye!\n");
            break;
        } else {
            printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

**Output:**

```
--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Display Total Salaries of All Employees
4. Search Employee by ID
5. Update Employee Data
6. Delete Employee Data
7. Exit
Enter your choice: 1
Enter Employee ID: 5678
Enter Name: MANGO
Enter Age: 22
Enter Department: AIDS
Enter Basic Salary: 45895
Enter Role (0 for SALES, 1 for TECHNICAL): 0
Enter Commission: 48795
Enter Sales Target: 812598
```

```
--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Display Total Salaries of All Employees
4. Search Employee by ID
5. Update Employee Data
6. Delete Employee Data
7. Exit
Enter your choice: 2

Employee ID: 5678
Name: MANGO
Age: 22
Department: AIDS
Basic Salary: 45895.00
Role: Sales
Commission: 48795.00
Sales Target: 812598.00
Total Salary: 94690.00
```

```
--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Display Total Salaries of All Employees
4. Search Employee by ID
5. Update Employee Data
6. Delete Employee Data
7. Exit
Enter your choice: 3
Employee ID: 5678, Total Salary: 94690.00
```

```
--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Display Total Salaries of All Employees
4. Search Employee by ID
5. Update Employee Data
6. Delete Employee Data
7. Exit
Enter your choice: 4
Enter Employee ID to search: 5678

Employee ID: 5678
Name: MANGO
Age: 22
Department: AIDS
Basic Salary: 45895.00
Role: Sales
Commission: 48795.00
Sales Target: 812598.00
Total Salary: 94690.00
```

```
--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Display Total Salaries of All Employees
4. Search Employee by ID
5. Update Employee Data
6. Delete Employee Data
7. Exit
Enter your choice: 5
Enter Employee ID to update: 5678
Updating data for Employee ID: 5678
Enter New Name: MANGO
Enter New Age: 25
Enter New Department: CCE
Enter New Basic Salary: 135674
Enter New Commission: 812654
Enter New Sales Target: 128974644
```

## Post Lab Subjective/Objective type Questions:

**1.** What is the difference between a structure and a union in C?

Ans: *The main difference between a structure and a union in C is the way they store their data. The key differences between a structure and a union in C are:*

- *A structure stores each member in separate memory locations, whereas a union stores all its members in the same memory location.*
- *The size of the structure is equal to or greater than the total size of all of its members. The size of the union is the size of its largest member.*
- *Structure is mainly used for storing various data types while union is mainly used for storing one of the many data types.*
- *In structure, you can retrieve any member at a time on the other hand in union; you can access one member at a time.*
- *Structure supports flexible array while union does not support a flexible array.*

## Conclusion:

In this module we learnt about structures and unions.

- A structure is a user-defined data type in C that groups variables of different types under a single name.

**Syntax:**

```
struct structure_name
{
data_type member1;
data_type member2;
```

*// more members*
*};*

- A union is a user-defined data type similar to a structure, but with one key difference: all members of a union share the same memory location.

**Syntax:**

*union union_name*
*{*
*data_type member1;*
*data_type member2;*
*// more members*
*};*

**Signature of faculty in-charge with Date:**