Course Name:	**Programming in C**	**Semester:**	**II**
Date of Performance:	**29/03/2025**	**DIV/ Batch No:**	**C2-2**
Student Name:	**Ashwera Hasan**	**Roll No:**	**16010124107**

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:

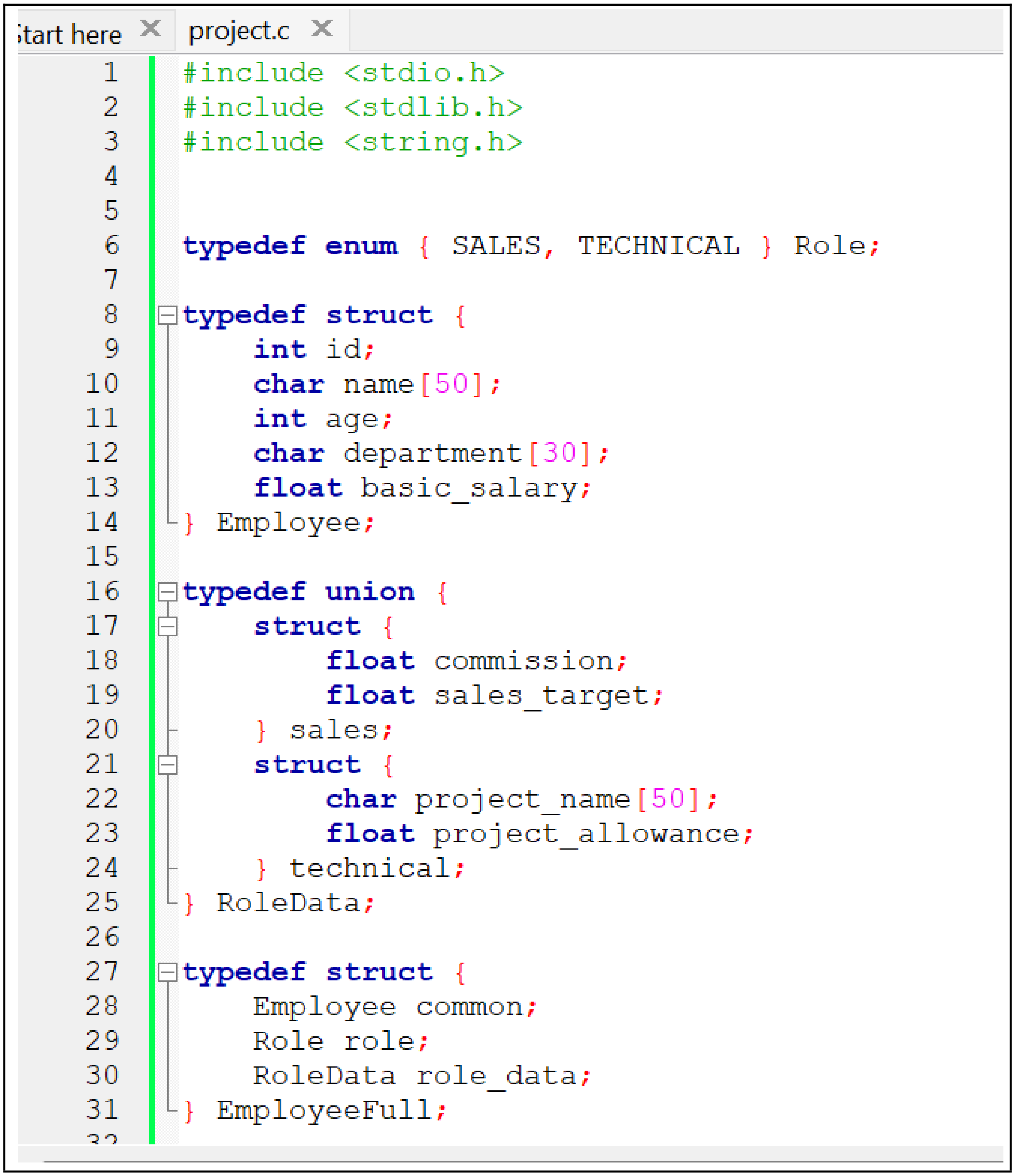
1. **Employee Data Input:**
   - Each employee should have the following common attributes:
     - Employee ID (integer)
     - Name (string)
     - Age (integer)
     - Department (string)
     - Basic Salary (float)
   - Depending on the employee's role, additional attributes should be stored:
     - 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:

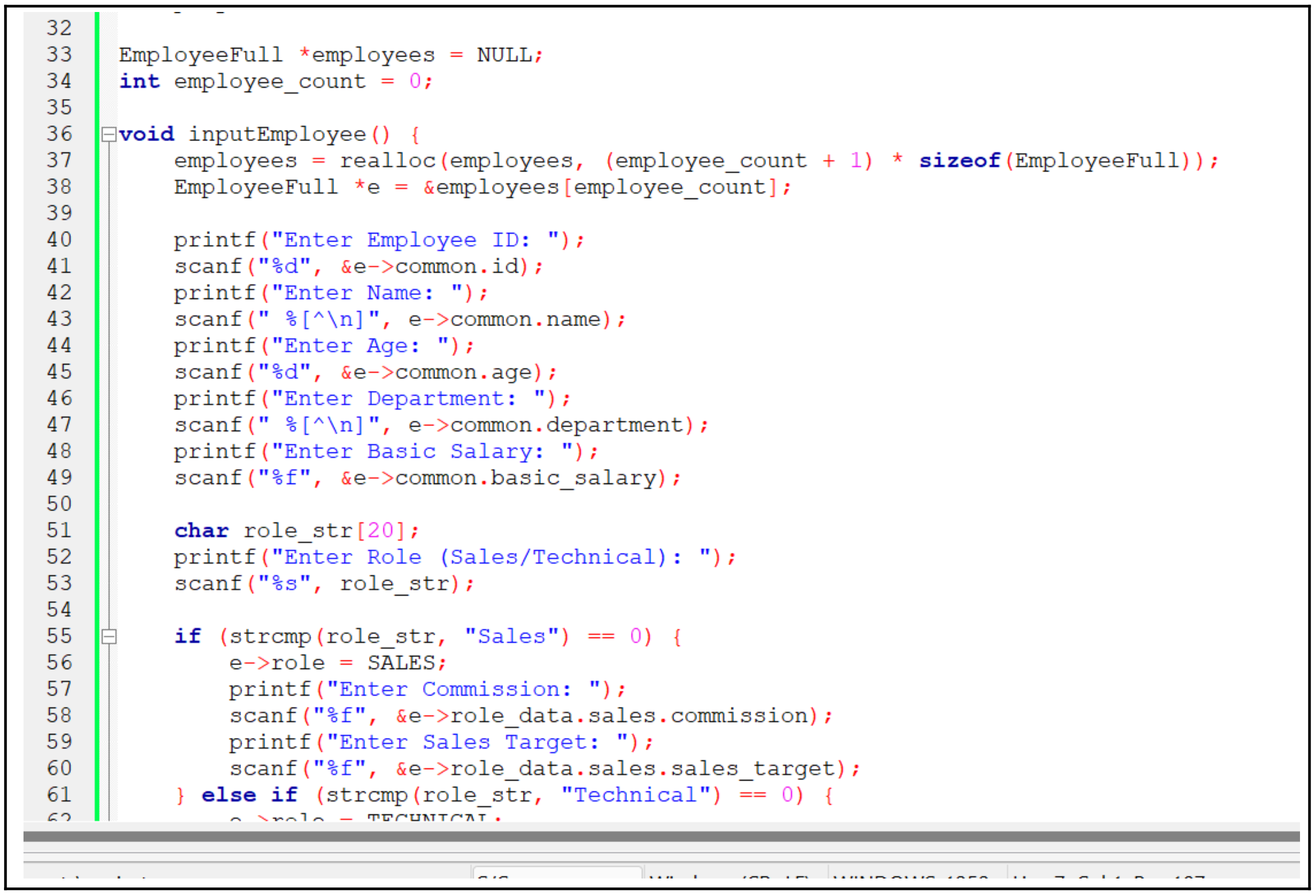
1. Use a **structure** to represent an employee with common attributes.
2. Use a **union** to store role-specific attributes (either for sales or technical employees).
3. Use an **enum** to differentiate between employee roles (e.g., SALES, TECHNICAL).

4. Implement dynamic memory allocation to store employee records. 5. Provide a menu-driven interface for the user to perform the above operations.

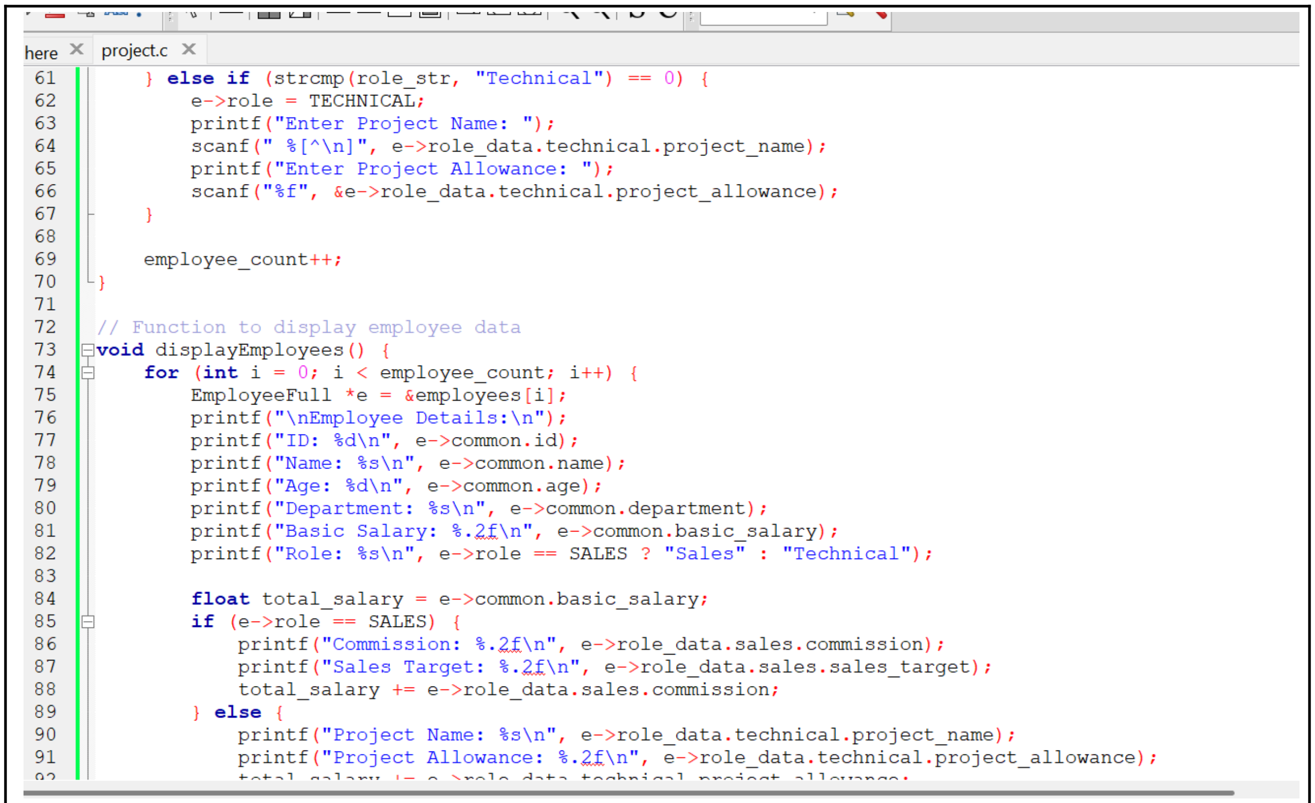
Code :


tart here × project.c ×

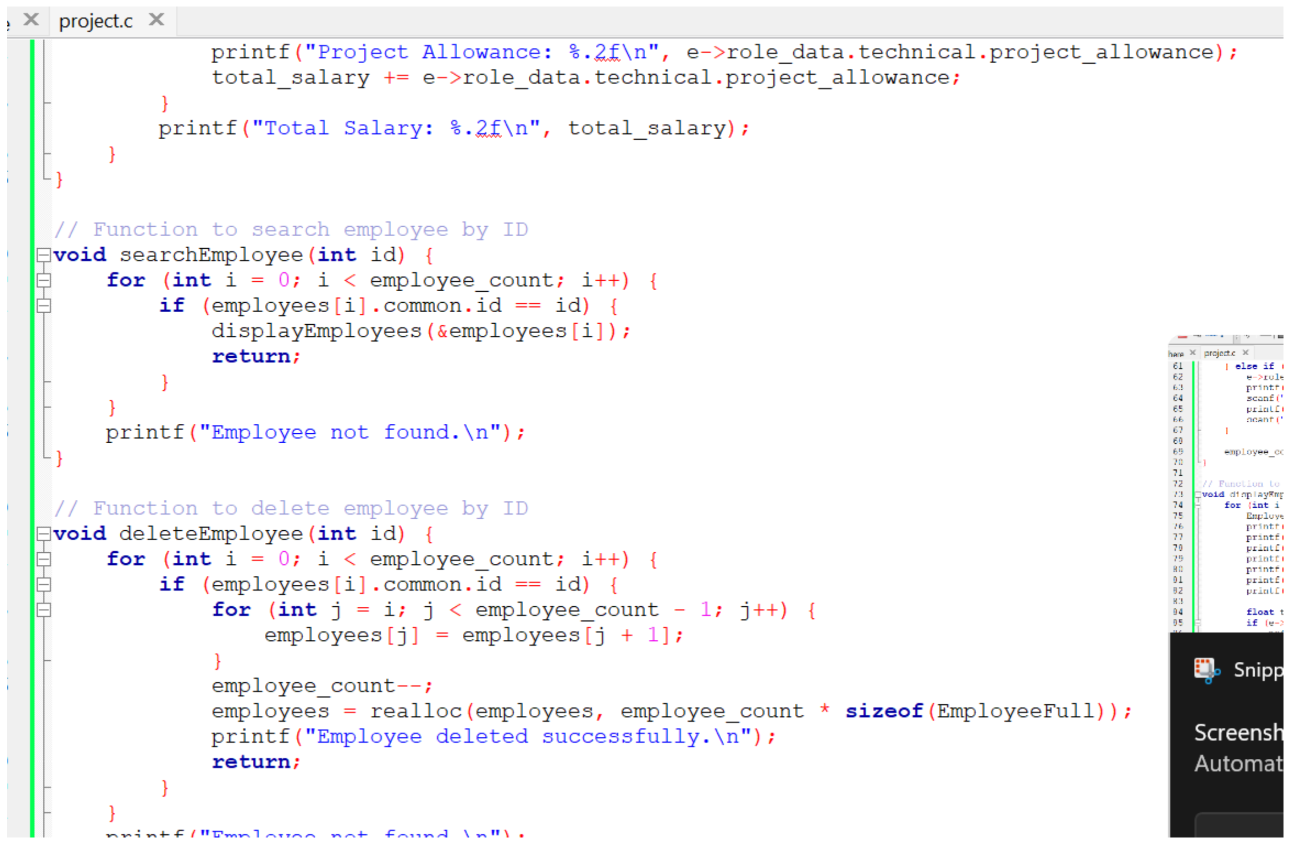
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


typedef enum { SALES, TECHNICAL } Role;

typedef struct {
    int id;
    char name[50];
    int age;
    char department[30];
    float basic_salary;
} Employee;

typedef union {
    struct {
        float commission;
        float sales_target;
    } sales;
    struct {
        char project_name[50];
        float project_allowance;
    } technical;
} RoleData;

typedef struct {
    Employee common;
    Role role;
    RoleData role_data;
} EmployeeFull;
```



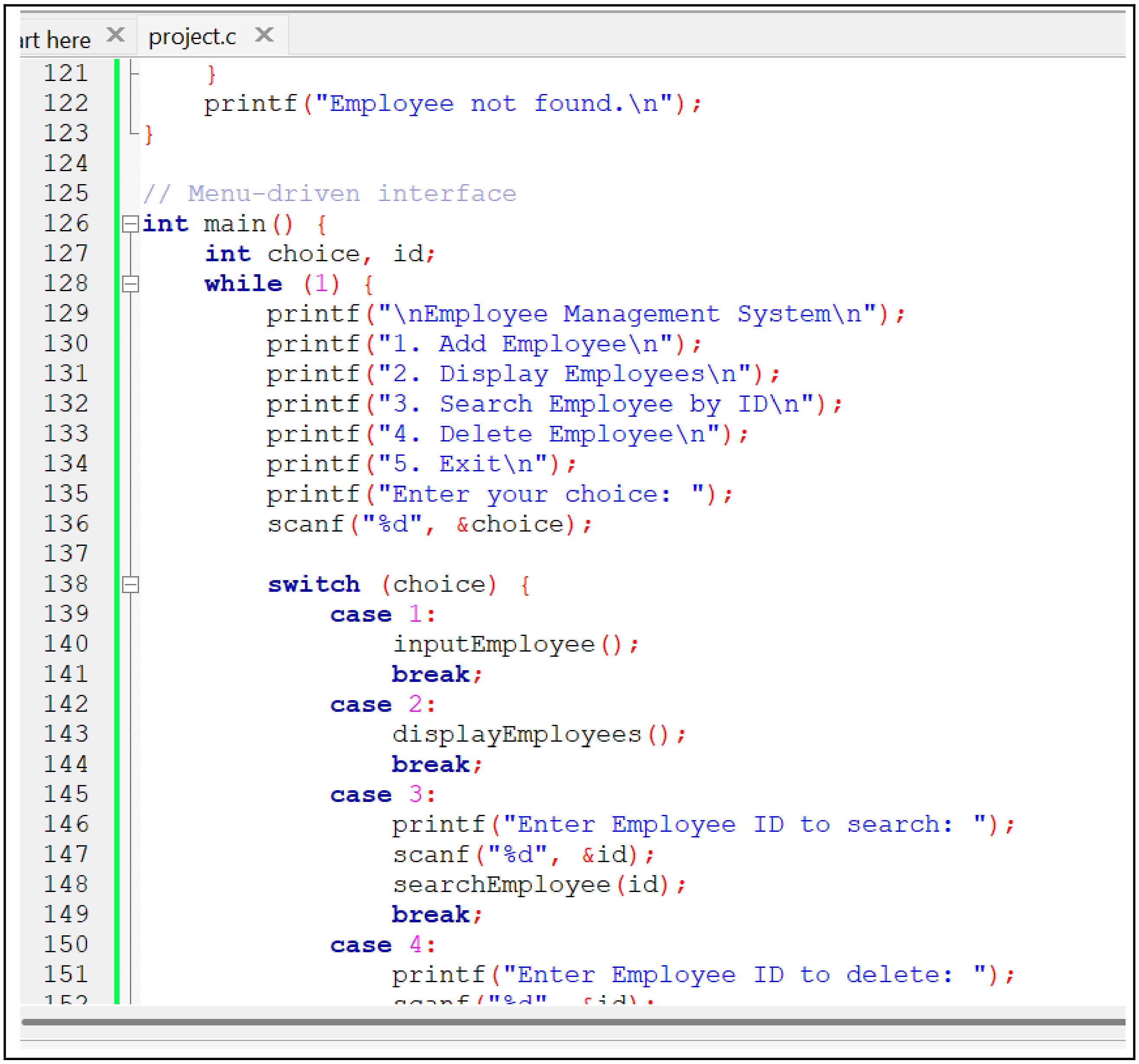
```c

EmployeeFull *employees = NULL;
int employee_count = 0;

void inputEmployee() {
    employees = realloc(employees, (employee_count + 1) * sizeof(EmployeeFull));
    EmployeeFull *e = &employees[employee_count];

    printf("Enter Employee ID: ");
    scanf("%d", &e->common.id);
    printf("Enter Name: ");
    scanf(" %[^\n]", e->common.name);
    printf("Enter Age: ");
    scanf("%d", &e->common.age);
    printf("Enter Department: ");
    scanf(" %[^\n]", e->common.department);
    printf("Enter Basic Salary: ");
    scanf("%f", &e->common.basic_salary);

    char role_str[20];
    printf("Enter Role (Sales/Technical): ");
    scanf("%s", role_str);

    if (strcmp(role_str, "Sales") == 0) {
        e->role = SALES;
        printf("Enter Commission: ");
        scanf("%f", &e->role_data.sales.commission);
        printf("Enter Sales Target: ");
        scanf("%f", &e->role_data.sales.sales_target);
    } else if (strcmp(role_str, "Technical") == 0) {
        e->role = TECHNICAL;
```



```c
    } else if (strcmp(role_str, "Technical") == 0) {
        e->role = TECHNICAL;
        printf("Enter Project Name: ");
        scanf(" %[^\n]", e->role_data.technical.project_name);
        printf("Enter Project Allowance: ");
        scanf("%f", &e->role_data.technical.project_allowance);
    }

    employee_count++;
}

// Function to display employee data
void displayEmployees() {
    for (int i = 0; i < employee_count; i++) {
        EmployeeFull *e = &employees[i];
        printf("\nEmployee Details:\n");
        printf("ID: %d\n", e->common.id);
        printf("Name: %s\n", e->common.name);
        printf("Age: %d\n", e->common.age);
        printf("Department: %s\n", e->common.department);
        printf("Basic Salary: %.2f\n", e->common.basic_salary);
        printf("Role: %s\n", e->role == SALES ? "Sales" : "Technical");

        float total_salary = e->common.basic_salary;
        if (e->role == SALES) {
            printf("Commission: %.2f\n", e->role_data.sales.commission);
            printf("Sales Target: %.2f\n", e->role_data.sales.sales_target);
            total_salary += e->role_data.sales.commission;
        } else {
            printf("Project Name: %s\n", e->role_data.technical.project_name);
            printf("Project Allowance: %.2f\n", e->role_data.technical.project_allowance);
            total_salary += e->role_data.technical.project_allowance;
```

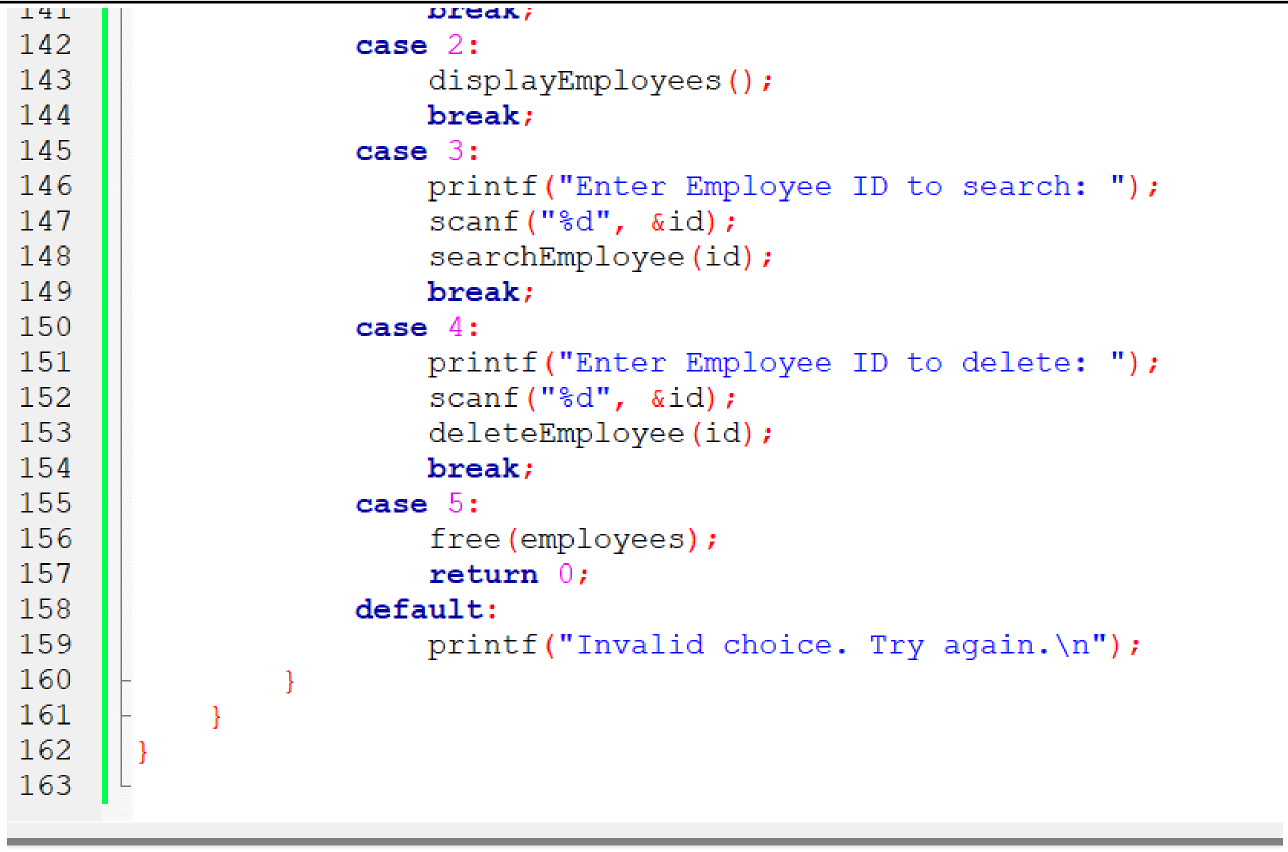


```c
            printf("Project Allowance: %.2f\n", e->role_data.technical.project_allowance);
            total_salary += e->role_data.technical.project_allowance;
        }
        printf("Total Salary: %.2f\n", total_salary);
    }
}

// Function to search employee by ID
void searchEmployee(int id) {
    for (int i = 0; i < employee_count; i++) {
        if (employees[i].common.id == id) {
            displayEmployees(&employees[i]);
            return;
        }
    }
    printf("Employee not found.\n");
}

// Function to delete employee by ID
void deleteEmployee(int id) {
    for (int i = 0; i < employee_count; i++) {
        if (employees[i].common.id == id) {
            for (int j = i; j < employee_count - 1; j++) {
                employees[j] = employees[j + 1];
            }
            employee_count--;
            employees = realloc(employees, employee_count * sizeof(EmployeeFull));
            printf("Employee deleted successfully.\n");
            return;
        }
    }
    printf("Employee not found.\n");
```



rt here ✕ project.c ✕

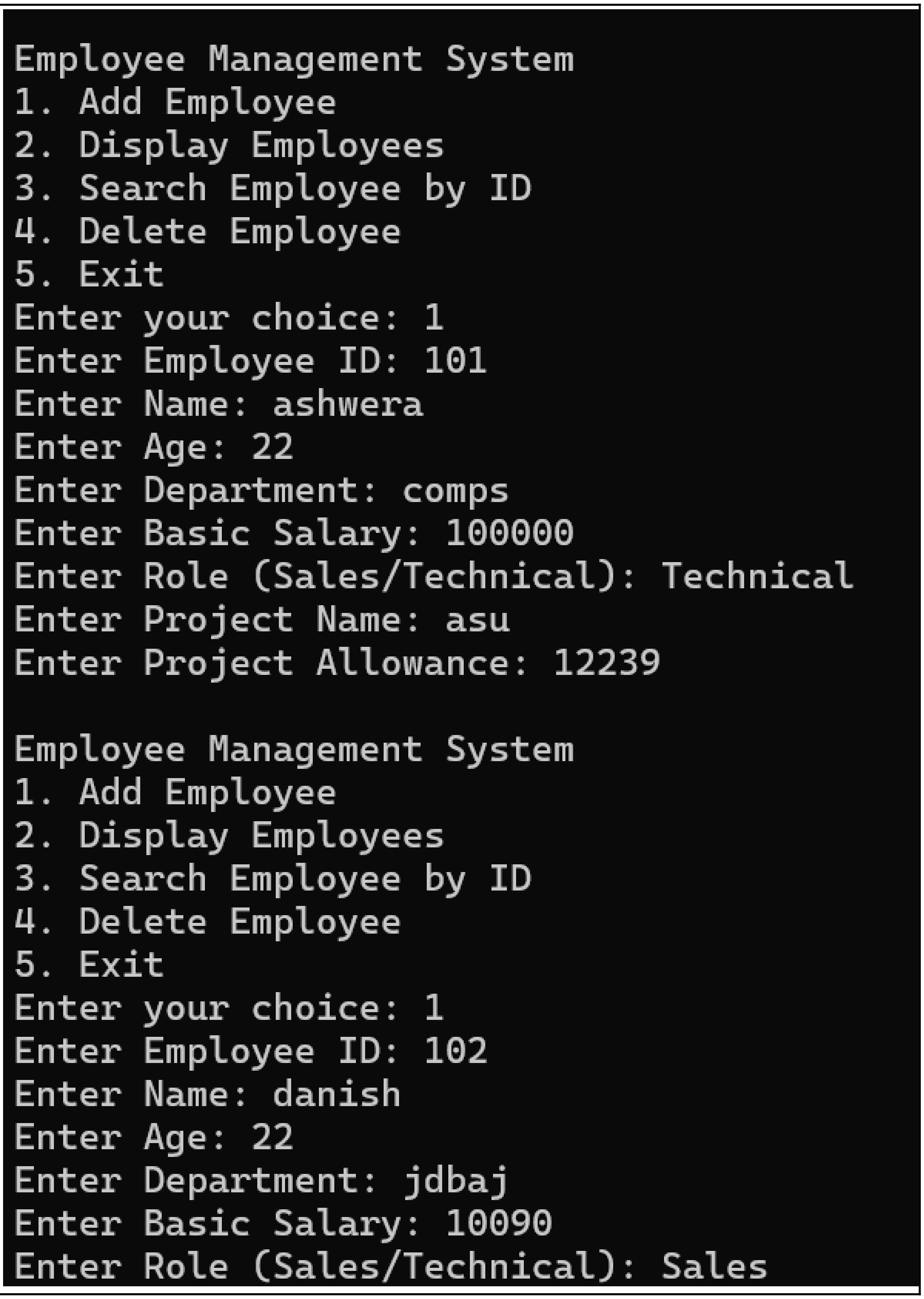
```
    }
    printf("Employee not found.\n");
}

// Menu-driven interface
int main() {
    int choice, id;
    while (1) {
        printf("\nEmployee Management System\n");
        printf("1. Add Employee\n");
        printf("2. Display Employees\n");
        printf("3. Search Employee by ID\n");
        printf("4. Delete Employee\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                inputEmployee();
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                printf("Enter Employee ID to search: ");
                scanf("%d", &id);
                searchEmployee(id);
                break;
            case 4:
                printf("Enter Employee ID to delete: ");
```



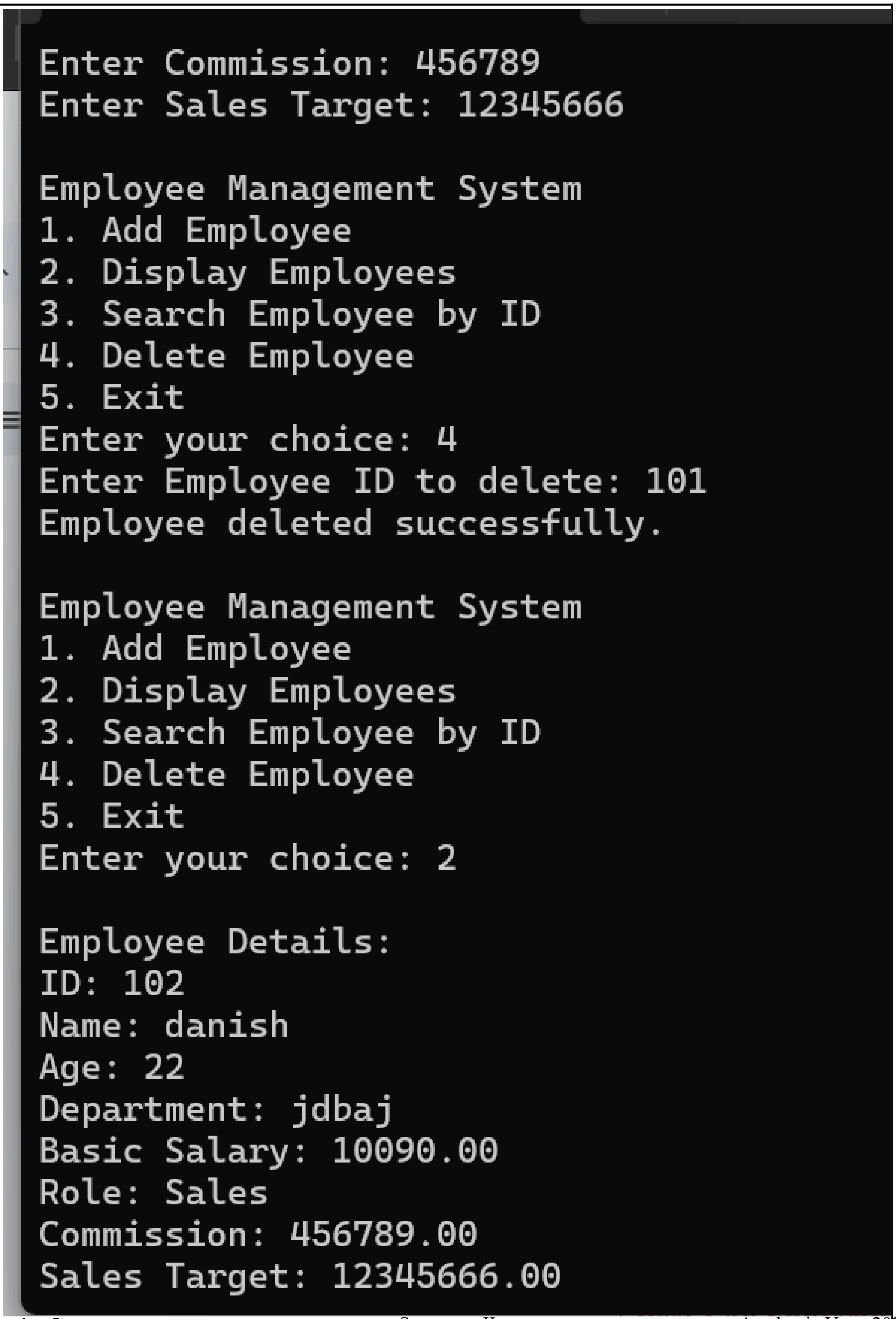
```
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                printf("Enter Employee ID to search: ");
                scanf("%d", &id);
                searchEmployee(id);
                break;
            case 4:
                printf("Enter Employee ID to delete: ");
                scanf("%d", &id);
                deleteEmployee(id);
                break;
            case 5:
                free(employees);
                return 0;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}
```

Output:

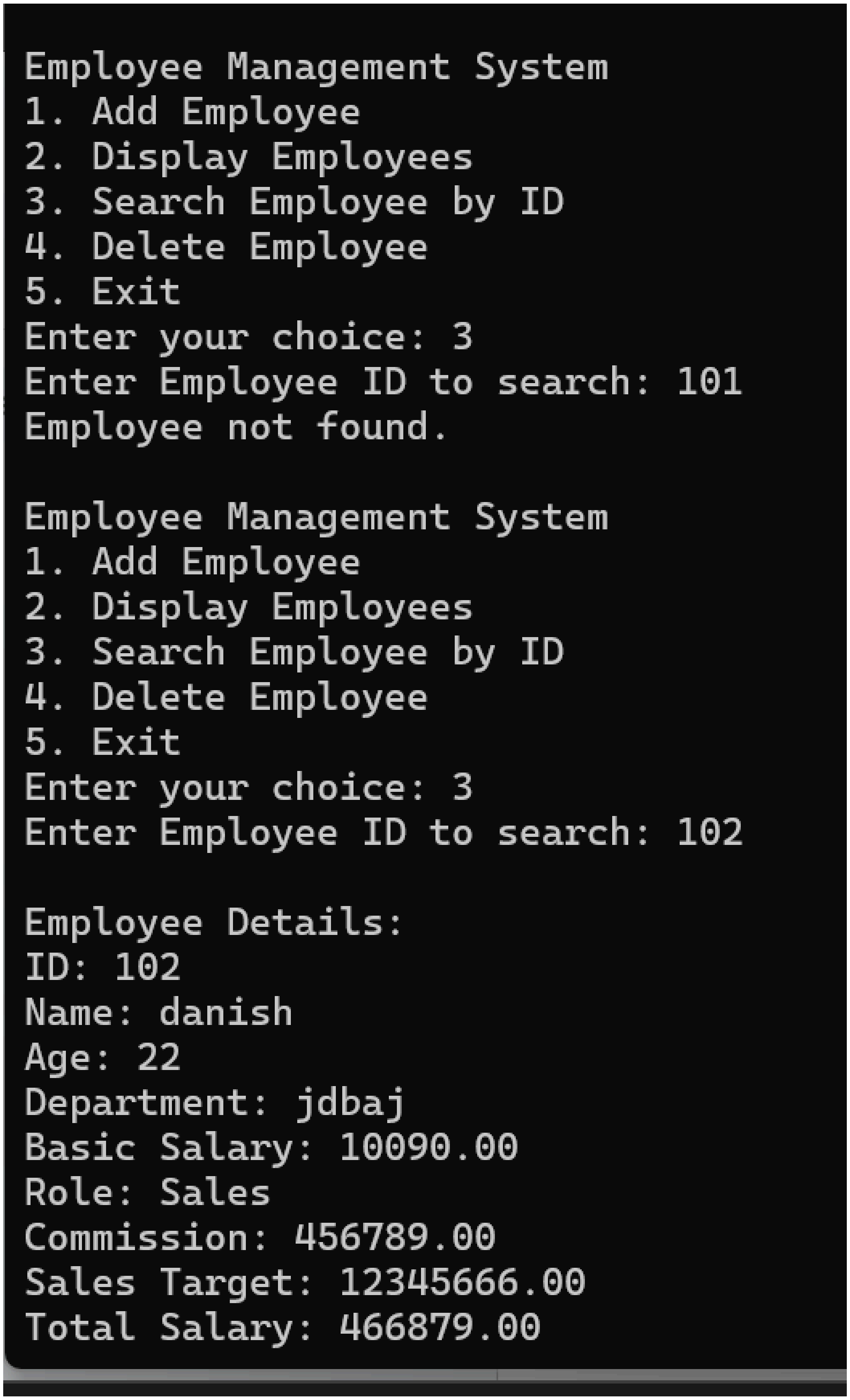
1.



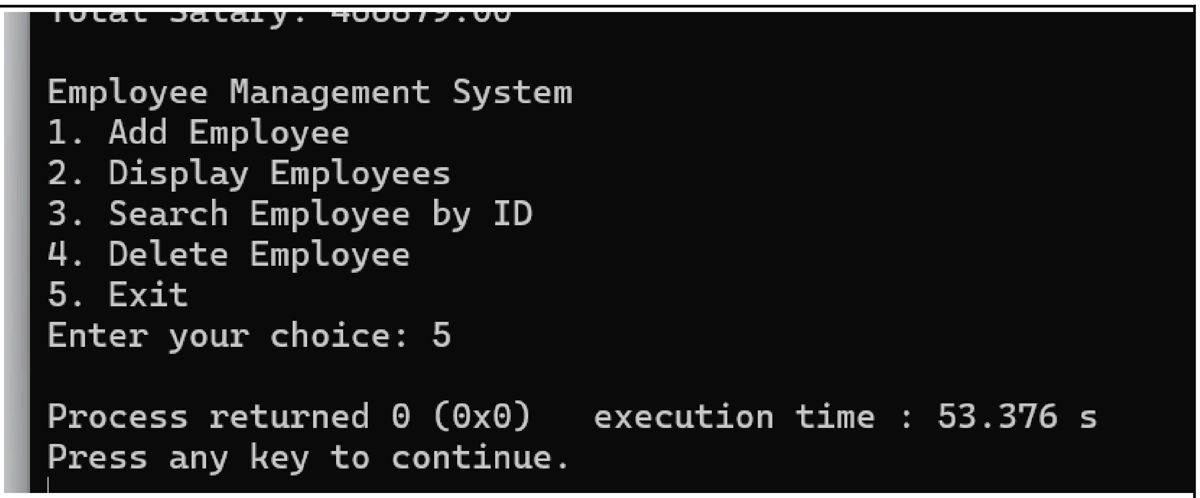
```
Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 1
Enter Employee ID: 101
Enter Name: ashwera
Enter Age: 22
Enter Department: comps
Enter Basic Salary: 100000
Enter Role (Sales/Technical): Technical
Enter Project Name: asu
Enter Project Allowance: 12239

Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 1
Enter Employee ID: 102
Enter Name: danish
Enter Age: 22
Enter Department: jdbaj
Enter Basic Salary: 10090
Enter Role (Sales/Technical): Sales
```



```
Enter Commission: 456789
Enter Sales Target: 12345666

Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 4
Enter Employee ID to delete: 101
Employee deleted successfully.

Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 2

Employee Details:
ID: 102
Name: danish
Age: 22
Department: jdbaj
Basic Salary: 10090.00
Role: Sales
Commission: 456789.00
Sales Target: 12345666.00
```



```
Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 3
Enter Employee ID to search: 101
Employee not found.

Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 3
Enter Employee ID to search: 102

Employee Details:
ID: 102
Name: danish
Age: 22
Department: jdbaj
Basic Salary: 10090.00
Role: Sales
Commission: 456789.00
Sales Target: 12345666.00
Total Salary: 466879.00
```



```
Employee Management System
1. Add Employee
2. Display Employees
3. Search Employee by ID
4. Delete Employee
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 53.376 s
Press any key to continue.
```

Post Lab Subjective/Objective type Questions:

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

   A structure in C allocates separate memory for each member. It allows simultaneous storage of different values. The size of a struct is the sum of the sizes of all members. Since all members are stored independently, no data overlap is noticed. Individual member can be accessed at a time.

   A union allocates the same memory that is shared by all members. This is done to prioritize memory efficiency. The size is equal to the size of the largest member. Data overlaps since the same memory is shared across many members. Only one member can be accessed at a time.

Conclusion:

We can use structs and unions in C to define compound data-types to suit our various purposes. Like in this program, we made use of the struct Employee to store all employee related data within one datatype. Further we used union RoleData to store the role-related info of all employees. Utilising these aspects in our code enhances its reusability and encapsulation and allows our code to be easily debugged.

Signature of faculty in-charge with Date: