|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Programming in C** | **Semester:** | **II** |
| **Date of Performance:** | **21 / 03 / 2025** | **DIV/ Batch No:** | **C4-1** |
| **Student Name:** | **Dhruv Pankhania** | **Roll No:** | **16010124216** |

**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  1. **Search Employee by ID:**    * Allow the user to search for an employee by their Employee ID and display their details. 2. **Update Employee Data:**    * Allow the user to update specific details of an employee (e.g., name, age, department, or role-specific data). 3. **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>  #include <string.h>  #define MAX\_EMP 1000  *enum* department  {    SALES,    TECHNICALS  };  *struct* sales  {  *float* commission;  *float* salesTarget;  };  *struct* technical  {  *char* projectName[1000];  *float* allowance;  };  *struct* employeeDetails  {  *int* id;  *char* name[1000];  *enum* department dept;  *int* age;  *float* basicSalary;  *struct*    {  *struct* sales s;  *struct* technical t;    } data;  };  *struct* employeeDetails emp[MAX\_EMP];  *int* empCount = 0;  *void* dataInput();  *void* displayData();  *void* calculateSalary();  *void* searchById();  *void* update();  *void* delete();  *int* main()  {  *int* flag = 1;    while (flag)    {  *int* choice;      printf("Enter 1 to Input Employee Data.\nEnter 2 to Display Employee Data.\nEnter 3 to Calculate a Employee`s Salary.\nEnter 4 to Search a Employee by ID.\nEnter 5 to Update a Employee Data.\nEnter 6 to Delete a Employee Data.\nEnter 7 to view all the option again.\nEnter 8 to exit.\nEnter your Choice: ");      scanf("%d", &choice);      switch (choice)      {      case 1:        dataInput();        break;      case 2:        displayData();        break;      case 3:        calculateSalary();        break;      case 4:        searchById();        break;      case 5:        update();        break;      case 6:        delete ();        break;      case 7:        flag = 1;        break;      case 8:        flag = 0;        break;      default:        printf("Please Enter a Valid Input.");        break;      }    }    return 0;  }  *void* dataInput()  {  *int* flag = 1;    while (flag && empCount < MAX\_EMP)    {    jump\_here:      printf("Enter Employee ID: ");      scanf("%d", &emp[empCount].id);      for (*int* i = 0; i < empCount; i++)      {        if (emp[empCount].id == emp[i].id)        {          printf("Employee ID already taken\n");          goto jump\_here;        }        else        {          break;        }      }      printf("Enter Employee Name: ");      scanf(" *%*[^\n]", &emp[empCount].name);      printf("Enter Employee Age: ");      scanf(" %d", &emp[empCount].age);      printf("Enter Employee`s Basic Salary: ");      scanf(" %f", &emp[empCount].basicSalary);  *int* dept;      printf("Enter 0 for Sales Department and 1 for Technical Department: ");      scanf("%d", &dept);      if (dept == 0)      {        emp[empCount].dept = SALES;        printf("Enter Employee`s Commission: ");        scanf("%f", &emp[empCount].data.s.commission);        printf("Enter Employee`s Sales Target: ");        scanf("%f", &emp[empCount].data.s.salesTarget);      }      else if (dept == 1)      {        emp[empCount].dept = TECHNICALS;        printf("Enter Employee`s Project Name: ");        scanf("%s", &emp[empCount].data.t.projectName);        printf("Enter Employee`s Allowance: ");        scanf("%f", &emp[empCount].data.t.allowance);      }      else      {        printf("Invalid Department!\n");        continue;      }      empCount++;      printf("Press 1 to input more Employees Data.\nEnter 0 to Exit.\n");      scanf("%d", &flag);    }  }  *void* displayData()  {    if (empCount == 0)    {      printf("No Employee Data Available\n");      return;    }    else    {      for (*int* i = 0; i < empCount; i++)      {        printf("Employee ID: %d\n", emp[i].id);        printf("Employee Name: %s\n", emp[i].name);        printf("Employee Age: %d\n", emp[i].age);        printf("Employee Basic Salary: %f\n", emp[i].basicSalary);        if (emp[i].dept == SALES)        {          printf("Department: Sales\n");          printf("Employee Commision: %f\n", emp[i].data.s.commission);          printf("Employee`s Sales Target: %f\n", emp[i].data.s.salesTarget);        }        else if (emp[i].dept == TECHNICALS)        {          printf("Department: Technical\n");          printf("Employee Project Name: %s\n", emp[i].data.t.projectName);          printf("Employee Allowance: %f\n", emp[i].data.t.allowance);        }      }    }  }  *void* calculateSalary()  {  *int* id;    printf("Enter Employee ID to Calculate Total Salary: ");    scanf("%d", &id);    for (*int* i = 0; i < empCount; i++)    {      if (emp[i].id == id)      {  *float* totalSalary = emp[i].basicSalary;        if (emp[i].dept == SALES)        {          totalSalary += emp[i].data.s.commission;        }        else        {          totalSalary += emp[i].data.t.allowance;        }        printf("The Total Salary of %s is %f\n", emp[i].name, totalSalary);        break;      }      else      {        printf("Enter Vaid Employee ID\n");      }    }  };  *void* searchById()  {  *int* id, found = 0;    printf("Enter Employee ID to be searched: ");    scanf("%d", &id);    for (*int* i = 0; i < empCount; i++)    {      if (emp[i].id == id)      {        printf("Employee ID: %d\n", emp[i].id);        printf("Employee Name: %s\n", emp[i].name);        printf("Employee Age: %d\n", emp[i].age);        printf("Employee Basic Salary: %f\n", emp[i].basicSalary);        if (emp[i].dept == SALES)        {          printf("Department: Sales");          printf("Employee Commision: %f\n", emp[i].data.s.commission);          printf("Employee Basic Salary: %f\n", emp[i].basicSalary);        }        else        {          printf("Department: Technical");          printf("Employee Project Name: %s\n", emp[i].data.t.projectName);          printf("Employee Allowance: %f\n", emp[i].data.t.allowance);        }      }      else      {        found = 1;      }    }    if (!found)    {      printf("Enter Valid Employee ID.\n");    }  }  *void* update()  {  *int* id, found = 0;    printf("Enter Employee ID to be Updated: ");    scanf("%d", &id);    for (*int* i = 0; i < empCount; i++)    {      if (emp[i].id == id)      {        printf("Enter New Employee ID: ");        scanf("%d", &emp[i].id);        printf("Enter New Employee Name: ");        scanf(" *%*[^\n]", &emp[i].name);        printf("Enter New Employee Age: ");        scanf(" %d", &emp[i].age);        printf("Enter New Employee`s Basic Salary: ");        scanf(" %f", &emp[i].basicSalary);  *int* dept;        printf("Enter 0 for Sales Department and 1 for Technical Department: ");        scanf("%d", &dept);        if (dept == 0)        {          emp[i].dept = SALES;          printf("Enter Employee`s Commission: ");          scanf("%f", &emp[i].data.s.commission);          printf("Enter Employee`s Sales Target: ");          scanf("%f", &emp[i].data.s.salesTarget);        }        else if (dept == 1)        {          printf("Enter Employee`s Project Name: ");          scanf(" *%*[^\n]", &emp[i].data.t.projectName);          printf("Enter Employee`s Allowance: ");          scanf("%f", &emp[i].data.t.allowance);        }        else        {          printf("Invalid Department!");          continue;        }        found = 1;      }    }    if (!found)    {      printf("Enter Valid Employee ID\n");    }  }  *void* delete()  {  *int* id;    printf("Enter Employee ID to be Deleted: ");    scanf("%d", &id);    for (*int* i = 0; i < empCount; i++)    {      if (emp[i].id == id)      {        emp[i].id = 0;        strcpy(emp[i].name, "");        emp[i].age = 0;        emp[i].basicSalary = 0;        if (emp[i].dept == SALES)        {          emp[i].data.s.commission = 0;          emp[i].data.s.salesTarget = 0;        }        else        {          strcpy(emp[i].data.t.projectName, "");          emp[i].data.t.allowance = 0;        }        empCount--;        printf("Employee Data Deleted Successfully\n");      }    }  } |

|  |
| --- |
| **Output:** |
|  |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. What is the difference between a structure and a union in C?   In C, a structure allocates memory for all its members individually, allowing multiple values to be stored and accessed simultaneously. The total size of a structure is the sum of the sizes of its members. In contrast, a union uses a shared memory space for all its members, with the size equal to its largest member. Since only one member can hold a value at a time, modifying one member overwrites the data of the others. Structures are useful for storing multiple data types together, while unions are ideal for saving memory when only one value is needed at a time. |

|  |
| --- |
| **Conclusion:** |
| We learned to use **struct**, **union**, and **enum** in C to manage employee data efficiently. We created an array of structs to store multiple employee records. The system allowed us to perform various operations like adding, displaying, updating, and deleting employee records, demonstrating practical applications of these concepts in real-world scenarios. |

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