CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

REPORT ON PROJECT BASED LEARNING

"EMPLOYEE INFORMATION USING ARRAY"

SUB: DATA STRUCTURE LAB

SUBMITTED BY:

STUDENT NAME: Divyam Bansal

BRANCH: BCA (DATA SCIENCE)

SUB CODE: 24CAP-152

UID: 24BCD10096

CLASS: 24BCD-2A

SUBMITTED To:

TEACHER NAME: Prof Jyoti Rani

Employee Management System

Index

1. Introduction

The Employee Management System is a simple C-based console application designed to manage employee records. It offers functionalities such as adding, displaying, searching, and deleting employee details, along with analyzing salary data department-wise.

Key functionalities include:

- **Adding New Employee Records** – Users can input and store essential employee information such as name, ID, department, and salary.
- **Displaying Employee Information** – A structured format displays all existing employee records for easy viewing and reference.
- **Searching for Specific Employees** – The system allows users to search for employees using unique identifiers like employee ID or name.
- **Deleting Records** – Unwanted or outdated employee records can be easily removed from the system.
- **Department-wise Salary Analysis** – The application includes a feature to analyze and compare salary distributions across different departments, aiding in human resource planning and budgeting.

2. Objective

The primary objective of this project is to:

- **Manage Employee Information Efficiently:**
  The system is designed to store, organize, and retrieve employee records in a structured format. This includes details such as employee name, ID, department, and salary
- **Provide Department-wise Salary Insights:**
  This functionality helps in identifying salary trends, planning departmental budgets, and making informed HR decisions.
- **Perform Record Operations like Search, Update, and Delete:**
  The system supports basic CRUD (Create, Read, Update, Delete) operations. Users can quickly:
  Search for employee records using identifiers like name or ID,
  Update existing employee information,
  Delete records that are outdated or no longer relevant.
- **Understand and Implement Dynamic Memory Allocation in C:**
  Through this project, learners get hands-on experience with dynamic memory management in C using functions like `malloc()`, `calloc()`, `realloc()`, and `free()`.
- **Demonstrate Structured Programming Concepts:**
  The project reinforces the practice of structured programming—breaking down the program into modular, reusable functions.

3. Working Methodology

This system uses structured programming in C and performs the following operations:

Add: Adds a new employee, automatically assigning a unique ID.
Display: Lists employees filtered by department.
Search: Finds employees by ID, name, or department.
Delete: Removes an employee by their ID.
Highest/Lowest Salary: Shows the employee with the highest or lowest salary in a department.

Key Concepts Used:

- **Dynamic memory allocation (malloc / realloc)**:
  Allows the program to allocate memory at runtime, enabling flexible storage for varying numbers of employee records.
- **Structs for data organization**:
  Structures (struct) group related employee attributes (like ID, name, department, salary) into a single, manageable data unit.
- **String and input handling (strcasecmp, scanf)**:
  Functions like scanf gather user input, while strcasecmp enables case-insensitive string comparisons for more flexible searches.

- **Menu-driven interface using loops and conditional statements**:
  Repeatedly presents user options and executes specific functions using loops (while, do-while) and conditionals (if, switch)

4. Source Code

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

struct stu {
   char nam[50];
   char dep[60];
   int id;
   double sal;
};
struct stu *emp = NULL;
int count = 0;
int next_id = 100;
char *gap = "\n\n\n";

// ----ADD FUNCTION----
void add() {
   struct stu *temp = realloc(emp, (count + 1) * sizeof(struct stu));
   if (temp == NULL) {
      printf("MEMORY ALLOCATION FAILED!\n");
      exit(1);
   }
   emp = temp;
   emp[count].id = next_id++;
   printf("----------ADDING EMPLOYEE----------\n\n");
   printf("ASSIGNED EMPLOYEE ID: %d\n", emp[count].id);

   printf("ENTER NAME: ");
   scanf(" %[^\n]", emp[count].nam); // Accepts names with spaces
   printf("ENTER DEPARTMENT (IT/SALES/FINANCE): ");
   scanf(" %[^\n]", emp[count].dep);
   printf("ENTER SALARY: ");
   scanf("%lf", &emp[count].sal);

   count++;
   printf("----- EMPLOYEE ADDED SUCCESSFULLY -----\n");
   printf("%s", gap);
}

// ----DISPLAY FUNCTION----
```

```
void display() {
  if (count == 0) {
    printf("NO EMPLOYEE RECORDS TO DISPLAY!\n");
    printf("%s", gap);
    return;
  }

  char dept[60];
  printf("ENTER DEPARTMENT TO DISPLAY EMPLOYEES: ");
  scanf(" %[^\n]", dept);
  int found = 0;

  printf("----------EMPLOYEE LIST (%s)----------\n\n", dept);
  for (int i = 0; i < count; i++) {
    if (strcasecmp(emp[i].dep, dept) == 0) {
      found = 1;
      printf("ID: %d\nNAME: %s\nDEPARTMENT: %s\nSALARY: %.2lf\n\n",
          emp[i].id, emp[i].nam, emp[i].dep, emp[i].sal);
    }
  }
  if (!found) printf("NO EMPLOYEES FOUND IN %s DEPARTMENT\n", dept);
  printf("%s", gap);
}

// ----SEARCH FUNCTION----
void search() {
  if (count == 0) {
    printf("NO EMPLOYEE RECORDS TO SEARCH!\n");
    printf("%s", gap);
    return;
  }

  int opt, search_id, found = 0;
  char search_str[60];

  printf("SEARCH BY:\n1. ID\n2. NAME\n3. DEPARTMENT\nEnter option: ");
  scanf("%d", &opt);

  if (opt == 1) {
    printf("ENTER ID: ");
    scanf("%d", &search_id);
    for (int i = 0; i < count; i++) {
      if (emp[i].id == search_id) {
        found = 1;
        printf("----------EMPLOYEE FOUND----------\n");
```

```
            printf("ID: %d\nNAME: %s\nDEPARTMENT: %s\nSALARY: %.2lf\n",
                emp[i].id, emp[i].nam, emp[i].dep, emp[i].sal);
            break;
          }
        }
      } else if (opt == 2) {
        printf("ENTER NAME: ");
        scanf(" %[^\n]", search_str);
        for (int i = 0; i < count; i++) {
          if (strcasecmp(emp[i].nam, search_str) == 0) {
            found = 1;
            printf("----------EMPLOYEE FOUND----------\n");
            printf("ID: %d\nNAME: %s\nDEPARTMENT: %s\nSALARY: %.2lf\n",
                emp[i].id, emp[i].nam, emp[i].dep, emp[i].sal);
            break;
          }
        }
      } else if (opt == 3) {
        printf("ENTER DEPARTMENT: ");
        scanf(" %[^\n]", search_str);
        for (int i = 0; i < count; i++) {
          if (strcasecmp(emp[i].dep, search_str) == 0) {
            found = 1;
            printf("----------EMPLOYEE FOUND----------\n");
            printf("ID: %d\nNAME: %s\nDEPARTMENT: %s\nSALARY: %.2lf\n",
                emp[i].id, emp[i].nam, emp[i].dep, emp[i].sal);
            break;
          }
        }
      } else {
        printf("INVALID OPTION!\n");
      }

      if (!found) printf("EMPLOYEE NOT FOUND!\n");
      printf("%s", gap);
    }

    // ----DELETE FUNCTION----
    void dlt() {
      if (count == 0) {
        printf("NO EMPLOYEE RECORDS TO DELETE!\n");
        printf("%s", gap);
        return;
      }
```

```
    int del_id, found = 0;
    printf("ENTER ID TO DELETE: ");
    scanf("%d", &del_id);

    for (int i = 0; i < count; i++) {
        if (emp[i].id == del_id) {
            found = 1;
            for (int j = i; j < count - 1; j++) {
                emp[j] = emp[j + 1];
            }
            count--;
            struct stu *temp = realloc(emp, count * sizeof(struct stu));
            if (temp != NULL || count == 0) { // realloc(0) may return NULL
                emp = temp;
            }
            printf("EMPLOYEE WITH ID %d DELETED.\n", del_id);
            break;
        }
    }
    if (!found) printf("EMPLOYEE WITH ID %d NOT FOUND!\n", del_id);
    printf("%s", gap);
}

// ----HIGHEST SALARY FUNCTION----
void highest() {
    if (count == 0) {
        printf("NO EMPLOYEE RECORDS TO EVALUATE HIGHEST SALARY!\n");
        printf("%s", gap);
        return;
    }

    char dpt[60];
    int found = 0, max = -1;
    printf("ENTER DEPARTMENT TO CHECK HIGHEST SALARY: ");
    scanf(" %[^\n]", dpt);

    for (int i = 0; i < count; i++) {
        if (strcasecmp(emp[i].dep, dpt) == 0) {
            if (!found || emp[i].sal > emp[max].sal) {
                max = i;
                found = 1;
            }
        }
    }
```

```
    if (found) {
        printf("EMPLOYEE WITH HIGHEST SALARY IN %s:\n", dpt);
        printf("ID: %d\nNAME: %s\nSALARY: %.2lf\n", emp[max].id, emp[max].nam,
emp[max].sal);
    } else {
        printf("NO EMPLOYEES FOUND IN DEPARTMENT %s\n", dpt);
    }
    printf("%s", gap);
}

// -----LOWEST SALARY FUNCTION----
void lowest() {
    if (count == 0) {
        printf("NO EMPLOYEE RECORDS TO EVALUATE LOWEST SALARY!\n");
        printf("%s", gap);
        return;
    }

    char dpt[60];
    int found = 0, min = -1;
    printf("ENTER DEPARTMENT TO CHECK LOWEST SALARY: ");
    scanf(" %[^\n]", dpt);

    for (int i = 0; i < count; i++) {
        if (strcasecmp(emp[i].dep, dpt) == 0) {
            if (!found || emp[i].sal < emp[min].sal) {
                min = i;
                found = 1;
            }
        }
    }

    if (found) {
        printf("EMPLOYEE WITH LOWEST SALARY IN %s:\n", dpt);
        printf("ID: %d\nNAME: %s\nSALARY: %.2lf\n", emp[min].id, emp[min].nam,
emp[min].sal);
    } else {
        printf("NO EMPLOYEES FOUND IN DEPARTMENT %s\n", dpt);
    }
    printf("%s", gap);
}

// ----MAIN FUNCTION----
int main() {
    int choice;
```

```
    printf("\t\t\t\t\t*~**~**~**~**~**~**~**~**~**~**~*\n");
    printf("\t\t\t\t\t*    EMPLOYEE MANAGEMENT SYSTEM    *\n");
    printf("\t\t\t\t\t*~**~**~**~**~**~**~**~**~**~**~*\n\n\n");

    do {
        printf("========== MENU ==========");
        printf("\n1. Add Employee\n2. Display Employees by Department\n3.
Search\n4. Delete\n5. Highest Salary by Department\n6. Lowest Salary by
Department\n7. Exit\n");
        printf("ENTER CHOICE: ");
        scanf("%d", &choice);
        printf("%s", gap);

        switch (choice) {
            case 1: add(); break;
            case 2: display(); break;
            case 3: search(); break;
            case 4: dlt(); break;
            case 5: highest(); break;
            case 6: lowest(); break;
            case 7: printf("EXITING...\n"); break;
            default: printf("INVALID CHOICE!\n");
        }
    } while (choice != 7);

    free(emp);
    return 0;
}
```

5. Scope For Enhancement

The following aspects can be adjusted as per requirement.

- **Add File I/O:**
  Integrate file handling (fopen, fread, fwrite) to save and load employee data across sessions.
- **Implement Update Feature:**
  Allow modification of employee records (e.g., change department, name, or salary).
- **Improve Input Handling:**
  Use fgets() instead of scanf("%s", ...) to accept full names and robust string input.
- **Add Input Validation:**
  Check for valid numerical entries, prevent empty or invalid strings, and ensure salaries are positive.
- **Search Enhancements:**
  Show all matching results, and allow partial or case-insensitive matches for names and departments.
- **Sort and Filter Functions:**
  Implement sorting employees by salary, name, or ID, and allow filtered views (e.g., top 5 highest paid).
- **User Interface Improvement:**
  Make the menu cleaner, introduce clear screen options, and provide a summary at the end of each operation.

6. Output

```
                                        *~**~**~**~**~**~**~**~**~**~**~**~**~**~*
                                        *     EMPLOYEE MANAGEMENT SYSTEM      *
                                        *~**~**~**~**~**~**~**~**~**~**~**~**~**~*


========== MENU ==========
1. Add Employee
2. Display Employees by Department
3. Search
4. Delete
5. Highest Salary by Department
6. Lowest Salary by Department
7. Exit
ENTER CHOICE: 1



----------ADDING EMPLOYEE----------

ASSIGNED EMPLOYEE ID: 100
ENTER NAME: Divyam Bansal
ENTER DEPARTMENT (IT/SALES/FINANCE): FINANCE
ENTER SALARY: 355000
----- EMPLOYEE ADDED SUCCESSFULLY -----
```

```
========== MENU ==========
1. Add Employee
2. Display Employees by Department
3. Search
4. Delete
5. Highest Salary by Department
6. Lowest Salary by Department
7. Exit
ENTER CHOICE: 3



SEARCH BY:
1. ID
2. NAME
3. DEPARTMENT
Enter option: 1
ENTER ID: 102
----------EMPLOYEE FOUND----------
ID: 102
NAME: Ramesh Suresh
DEPARTMENT: SALES
SALARY: 277000.00
```

```
ENTER DEPARTMENT TO CHECK LOWEST SALARY: IT
EMPLOYEE WITH LOWEST SALARY IN IT:
ID: 101
NAME: Tony Stark
SALARY: 455000.00
```

```
EXITING...


...Program finished with exit code 0
Press ENTER to exit console.
```