

BATCH-68

BTECH CSE

TEAM MEMBERS:

KALYAN RAM[590027112]

NAVEEN KUMAR[590025871]

C-PROJECT

EMPLOYEE SALARY MANAGEMENT SYSTEM

REPORT

1. Problem definition

Create a small console-based Employee Salary Management System in C that can:

- Store employee records (ID, name, basic salary, allowances, deductions).
- Calculate gross salary, net salary and optionally tax.
- Add, view, update, and delete employee records (simple file-based persistence).
- Display a payslip for a single employee.

This project is aimed at demonstrating modular C programming using a header file and implementation file, simple file I/O for persistence, and a menu-driven interface.

2. Flowchart (ASCII)

Start

|

v

Show Menu ---> [1] Add Employee

- | [2] View Employee
- | [3] Update Employee
- | [4] Delete Employee
- | [5] List All
- | [6] Exit

v

Get choice

|
+--> If 1: Input details -> Save to file -> Confirm -> Back to
Menu

|
+--> If 2: Enter ID -> Read file -> Display payslip -> Back
to Menu

|
+--> If 3: Enter ID -> Read file -> Modify record -> Save ->
Back to Menu

|
+--> If 4: Enter ID -> Remove from file -> Back to Menu

|
+--> If 5: Read all -> Display list -> Back to Menu

+--> If 6: Exit

v

End

3. Algorithm (high-level)

1. Start program and open (or create) data file employees.dat in binary append/read mode when needed.
2. Present menu to user.
3. For Add Employee: Get employee details, compute derived fields (gross, net), append record to file.
4. For View Employee: Scan records for the given ID and show employee's payslip.
5. For Update Employee: Read all records, modify the matching record, write back updated list (temporary file technique).
6. For Delete Employee: Same as update but exclude the matching record when rewriting file.
7. For List All: Read and print brief information for each record.
8. Loop until user chooses Exit.
9. Close file and terminate.

4. Problems faced by the group (2 members)

Member A: Implementing safe file update (atomic rewrite) and ensuring no data loss when updating/deleting records.

Member B: Designing clear input validation for salary fields and formatting the payslip output uniformly.

Work distribution suggestion:

Member A: File I/O, record storage format, update/delete functions.

Member B: User interface, validation, payslip formatting, testing.

5.Code:

Main.cpp

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

#define DATAFILE "employees.dat"

// --- Helper Functions ---

// Reads an integer and cleans the input buffer
int input_int(const char *prompt) {
```

```
int val;

printf("%s", prompt);

if (scanf("%d", &val) != 1) {
    // Invalid input, clear buffer and return a default or
    error value

    while (getchar() != '\n');

    return -1; // Indicate error
}

// consume remaining newline or characters in buffer

while (getchar() != '\n');

return val;
}

// Reads a string, removing the newline character

void input_string(const char *prompt, char *buffer, int size)
{
    printf("%s", prompt);

    if (fgets(buffer, size, stdin) != NULL) {
        size_t len = strlen(buffer);

        if (len > 0 && buffer[len - 1] == '\n') {

            buffer[len - 1] = '\0';

        }
    }
}
```

```
}
```

```
// --- Implementations of Functions from salary.h ---
```

```
void compute_salary(Employee *e) {  
    e->gross = e->basic + e->allowances;  
    e->net = e->gross - e->deductions;  
}
```

```
bool add_employee(const Employee *e, const char *filename)  
{  
    FILE *file = fopen(filename, "ab"); // Append binary  
    if (!file) return false;  
    if (fwrite(e, sizeof(Employee), 1, file) != 1) {  
        fclose(file);  
        return false;  
    }  
    fclose(file);  
    return true;  
}
```

```
bool find_employee(int id, Employee *out, const char  
*filename) {
```

```
FILE *file = fopen(filename, "rb");
if (!file) return false;
Employee temp;
while (fread(&temp, sizeof(Employee), 1, file) == 1) {
    if (temp.id == id) {
        *out = temp;
        fclose(file);
        return true;
    }
}
fclose(file);
return false;
}
```

```
bool update_employee(const Employee *e, const char
*filename) {
    FILE *file = fopen(filename, "r+b");
    if (!file) return false;
    Employee temp;
    long pos;
    while ((pos = ftell(file)) != -1 && fread(&temp,
sizeof(Employee), 1, file) == 1) {
        if (temp.id == e->id) {
```

```
fseek(file, pos, SEEK_SET);
if (fwrite(e, sizeof(Employee), 1, file) != 1) {
    fclose(file);
    return false;
}
fclose(file);
return true;
}
fclose(file);
return false;
}
```

```
bool delete_employee(int id, const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (!file) return false;
    FILE *temp_file = fopen("temp.dat", "wb");
    if (!temp_file) {
        fclose(file);
        return false;
    }
    Employee temp;
```

```
bool found = false;

while (fread(&temp, sizeof(Employee), 1, file) == 1) {

    if (temp.id != id) {

        fwrite(&temp, sizeof(Employee), 1, temp_file);

    } else {

        found = true;

    }

}

fclose(file);

fclose(temp_file);

if (found) {

    remove(filename);

    rename("temp.dat", filename);

} else {

    remove("temp.dat");

}

return found;

}
```

```
void list_employees(const char *filename) {

    FILE *file = fopen(filename, "rb");

    if (!file) {
```

```
    printf("No employees found or file error.\n");
    return;
}

Employee e;
printf("\n--- All Employees ---\n");
while (fread(&e, sizeof(Employee), 1, file) == 1) {
    printf("ID: %d, Name: %s, Basic: %.2f, Allowances: %.2f,
Deductions: %.2f, Gross: %.2f, Net: %.2f\n",
        e.id, e.name, e.basic, e.allowances, e.deductions,
        e.gross, e.net);
}
fclose(file);
}
```

// --- Menu Actions ---

```
void do_add() {
    Employee e;
    printf("\n--- Add New Employee ---\n");
    e.id = input_int("Enter ID: ");
    if (e.id == -1) {
        printf("Invalid ID entered.\n");
        return;
    }
```

```
}
```

```
// Check if ID already exists  
Employee dummy;  
if (find_employee(e.id, &dummy, DATAFILE)) {  
    printf("Error: Employee with ID %d already exists.\n",  
e.id);  
    return;  
}  
  
input_string("Enter Name: ", e.name, NAME_LEN);  
  
printf("Enter Basic Salary: ");  
if (scanf("%lf", &e.basic) != 1) {  
    printf("Invalid input for basic salary.\n");  
    while (getchar() != '\n');  
    return;  
}  
printf("Enter Allowances: ");  
if (scanf("%lf", &e.allowances) != 1) {  
    printf("Invalid input for allowances.\n");  
    while (getchar() != '\n');
```

```
    return;
}

printf("Enter Deductions: ");
if (scanf("%lf", &e.deductions) != 1) {
    printf("Invalid input for deductions.\n");
    while (getchar() != '\n');
    return;
}
while (getchar() != '\n'); // clear buffer after double inputs

compute_salary(&e);

if (add_employee(&e, DATAFILE)) {
    printf("Employee added successfully.\n");
} else {
    printf("Failed to add employee.\n");
}
}

void do_view() {
    int id = input_int("Enter employee ID to view: ");
    if (id == -1) {
```

```
    printf("Invalid ID entered.\n");
    return;
}

Employee e;
if (find_employee(id, &e, DATAFILE)) {
    printf("\n==== PAYSLIP ===\n");
    printf("ID:      %d\n", e.id);
    printf("Name:     %s\n", e.name);
    printf("Basic:    %.2f\n", e.basic);
    printf("Allowances: %.2f\n", e.allowances);
    printf("Deductions: %.2f\n", e.deductions);
    printf("-----\n");
    printf("GROSS:    %.2f\n", e.gross);
    printf("NET SALARY: %.2f\n", e.net);
    printf("=====*\n");
} else {
    printf("Employee not found.\n");
}
}

void do_update() {
    int id = input_int("Enter employee ID to update: ");
}
```

```
if (id == -1) {
    printf("Invalid ID entered.\n");
    return;
}

Employee e;
if (!find_employee(id, &e, DATAFILE)) {
    printf("Employee not found.\n");
    return;
}

printf("\nUpdating employee %s (ID %d)\n", e.name, e.id);

// Handle name update (allow blank to keep existing)
char name_buffer[NAME_LEN];
input_string("Enter new name (leave blank to keep): ",
name_buffer, NAME_LEN);
if (strlen(name_buffer) > 0) {
    strcpy(e.name, name_buffer);
}

// Handle numeric updates
printf("Enter new basic salary (current %.2f): ", e.basic);
```

```
double temp_d;  
if (scanf("%lf", &temp_d) == 1) e.basic = temp_d;  
  
printf("Enter new allowances (current %.2f): ",  
e.allowances);  
if (scanf("%lf", &temp_d) == 1) e.allowances = temp_d;  
  
printf("Enter new deductions (current %.2f): ",  
e.deductions);  
if (scanf("%lf", &temp_d) == 1) e.deductions = temp_d;  
  
while (getchar() != '\n'); // clear buffer  
  
compute_salary(&e);  
  
if (update_employee(&e, DATAFILE))  
    printf("Employee updated successfully.\n");  
else  
    printf("Failed to update.\n");  
}  
  
void do_delete() {  
    int id = input_int("Enter employee ID to delete: ");
```

```
if (id == -1) {  
    printf("Invalid ID entered.\n");  
    return;  
}  
  
if (delete_employee(id, DATAFILE))  
    printf("Employee deleted.\n");  
else  
    printf("Failed to delete (ID not found?).\n");  
}
```

```
void do_list() {  
    list_employees(DATAFILE);  
}
```

```
int main(void) {  
    int choice;  
    do {  
        printf("\nEmployee Salary Management\n");  
        printf("1. Add Employee\n");  
        printf("2. View Employee / Payslip\n");  
        printf("3. Update Employee\n");  
        printf("4. Delete Employee\n");
```

```
printf("5. List All Employees\n");
printf("6. Exit\n");

choice = input_int("Choose an option: ");
if (choice == -1) {
    printf("Invalid choice.\n");
    continue;
}

switch (choice) {
    case 1: do_add(); break;
    case 2: do_view(); break;
    case 3: do_update(); break;
    case 4: do_delete(); break;
    case 5: do_list(); break;
    case 6: printf("Exiting...\n"); break;
    default: printf("Invalid option.\n");
}

} while (choice != 6);

return 0;
```

Salary.c

```
#include <stdio.h> // Added for FILE and printf (good
practice, even if salary.h includes it)

#include <stdlib.h>

#include <string.h>

#include "salary.h"

// Compute gross and net.

// Gross = basic + allowances; Net = gross - deductions

void compute_salary(Employee *e) {

    e->gross = e->basic + e->allowances;

    e->net = e->gross - e->deductions;

}

bool add_employee(const Employee *e, const char *filename)
{
    FILE *fp = fopen(filename, "ab");

    if (!fp) return false;

    if (fwrite(e, sizeof(Employee), 1, fp) != 1) {

        fclose(fp);

        return false;
    }
}
```

```
fclose(fp);

return true;

}

bool find_employee(int id, Employee *out, const char
*filename) {

FILE *fp = fopen(filename, "rb");
if (!fp) return false;

Employee tmp;
bool found = false;
while (fread(&tmp, sizeof(Employee), 1, fp) == 1) {
    if (tmp.id == id) {
        *out = tmp;
        found = true;
        break;
    }
}
fclose(fp);
return found;
}
```

```
bool update_employee(const Employee *e, const char
*filename) {

    FILE *fp = fopen(filename, "rb");
    if (!fp) return false;

    FILE *tf = fopen("temp.dat", "wb");
    if (!tf) {
        fclose(fp);
        return false;
    }

    Employee tmp;
    bool found = false;

    while (fread(&tmp, sizeof(Employee), 1, fp) == 1) {
        if (tmp.id == e->id) {
            // Write the new (updated) employee data
            fwrite(e, sizeof(Employee), 1, tf);
            found = true;
        } else {
            // Write existing data
            fwrite(&tmp, sizeof(Employee), 1, tf);
        }
    }

    fclose(fp);
    fclose(tf);

    return found;
}
```

```
        }

    }

fclose(fp);
fclose(tf);

if (found) {
    remove(filename);
    rename("temp.dat", filename);
    return true;
} else {
    remove("temp.dat");
    return false;
}

bool delete_employee(int id, const char *filename) {
    FILE *fp = fopen(filename, "rb");
    if (!fp) return false;

    FILE *tf = fopen("temp.dat", "wb");
    if (!tf) {
```

```
fclose(fp);  
return false;  
}
```

```
Employee tmp;  
bool found = false;
```

```
while (fread(&tmp, sizeof(Employee), 1, fp) == 1) {  
    if (tmp.id == id) {  
        found = true;  
        // Do not write this record to temp file (effectively  
        deleting it)
```

```
} else {  
    fwrite(&tmp, sizeof(Employee), 1, tf);  
}  
}
```

```
fclose(fp);  
fclose(tf);
```

```
if (found) {  
    remove(filename);
```

```
    rename("temp.dat", filename);

    return true;

} else {

    remove("temp.dat");

    return false;

}

}
```

```
void list_employees(const char *filename) {

    FILE *fp = fopen(filename, "rb");

    if (!fp) {

        printf("No data found or file error.\n");

        return;

    }

}
```

```
Employee e;

printf("\n%-5s %-20s %-10s %-10s %-10s %-10s\n",
    "ID", "Name", "Basic", "Allow", "Deduct", "Gross",
    "Net");

printf("-----\n");

while (fread(&e, sizeof(Employee), 1, fp) == 1) {
```

```

        printf("%-5d %-20s %-10.2f %-10.2f %-10.2f %-10.2f %-
10.2f\n",
e.id, e.name, e.basic, e.allowances, e.deductions,
e.gross, e.net);

    }

printf("-----
-----\n");

fclose(fp);

}

```

Salary.h

```

main2.cpp salaryy.c salaryyy.h
1  ifndef SALARY_H
2  define SALARY_H
3
4  include <stdio.h>
5  include <stdbool.h>
6  include <string.h> // Added for potential string operations in implementations
7
8  define NAME_LEN 50
9
10 struct {
11     int id;
12     char name[NAME_LEN];
13     double basic;
14     double allowances; // Other allowances
15     double deductions; // Other deductions
16     double gross; // Computed gross salary
17     double net; // Computed net salary
18 } Employee;
19
20 // File operations
21 bool add_employee(const Employee *e, const char *filename);
22 bool find_employee(int id, Employee *out, const char *filename);
23 bool update_employee(const Employee *e, const char *filename);
24 bool delete_employee(int id, const char *filename);
25 void list_employees(const char *filename);
26
27 // Calculations
28 void compute_salary(Employee *e);
29
30 #endif // SALARY_H
31

```

OUTPUT:

```

Enter Deductions: 200000
Employee added successfully.

Employee Salary Management
1. Add Employee
2. View Employee / Payslip
3. Update Employee
4. Delete Employee
5. List All Employees
6. Exit
Choose an option: 5

--- All Employees ---
ID: 590027112, Name: Kalyan, Basic: 10000000.00, Allowances: 5000000.00, Deductions: 200000.00, Gross: 15000000.00, Net: 14800000.00
ID: 33000, Name: max_verstappen, Basic: 75000000.00, Allowances: 5000000.00, Deductions: 2000000.00, Gross: 80000000.00, Net: 78000000.00
ID: 78990, Name: praneeth, Basic: 20000000.00, Allowances: 350000.00, Deductions: 200000.00, Gross: 20350000.00, Net: 20150000.00
ID: 987, Name: praneeth, Basic: 20000000.00, Allowances: 300000.00, Deductions: 200000.00, Gross: 20300000.00, Net: 20100000.00

Employee Salary Management
1. Add Employee
2. View Employee / Payslip
3. Update Employee
4. Delete Employee
5. List All Employees
6. Exit
Choose an option: 6
Exiting...

Process exited after 55.77 seconds with return value 0
Press any key to continue . . .

```

7. Notes & enhancements

- Right now records are stored in a simple binary file. For larger projects use a database or CSV.
- Add input validation (non-negative salaries, unique ID check before add).
- Add search by name and sorting when listing.
- Add tax calculation and more complex salary components.