Employee Management System Report

Description of the Program

The Employee Management System is a console-based application written in C that allows users to manage a list of employees. The program provides functionality such as adding, deleting, modifying, searching, printing, and saving employee records. The system uses a struct to represent employee data, including information such as ID, name, salary, birth date, address, phone number, employment date, and email.

Sample Runs

Sample Run 1: Loading Existing Data

```
Enter the Filename to load from [if exists]: employees.txt

Loaded 10 Employees.

Good Morning!

Menu:

1. Add Employee

2. Delete Employee

3. Modify Employee

4. Search Employee

5. Print Employees

6. Save Employees to File

7. Quit

Enter your choice (1-7): 5

Selected: Print Employees

Sort by (N)ame, (D)ate of Birth, or (S)alary: N

12345, Name: John Doe, Salary: $50000, Birth Date: 10-05-1990, Address: 123 Main St, Phone: 555-1234, Employment Date: 01-01-2022, Email: john.doe@example.com

67890, Name: Jane Smith, Salary: $60000, Birth Date: 15-08-1985, Address: 456 Oak St, Phone: 555-5678, Employment Date: 02-01-2022, Email: jane.smith@example.com

...
```

Sample Run 2: Adding a New Employee

```
Menu:

1. Add Employee

2. Delete Employee

3. Modify Employee

4. Search Employee

5. Print Employees

6. Save Employees to File

7. Quit

Enter your choice (1-7): 1

Selected: Add Employee


Enter the number of Employees to be added: 1


Employee #11:


Enter the ID of the employee: 11111

Enter the name: Michael Jordan

Enter employee's salary : 75000

Enter Date of birth (DD-MM-YYYY): 17-02-1963

Enter the employee's address: 789 High St

Enter the employee's phone number: 01200000000

Enter employee's email address: michael.jordan@example.com
```

User Manual

1. **Loading Existing Data:**
   - When the program starts, it prompts the user to enter the filename to load employee data from. If the file exists, it loads the data into the system.
2. **Adding an Employee:**
   - Choose option 1 from the main menu to add an employee.
   - The program will ask for the number of employees to add. Enter an integer value.
   - For each employee, provide details such as ID, name, salary, birth date, address, phone number, and email.
3. **Deleting an Employee:**
   - Choose option 2 from the main menu to delete an employee.
   - Enter the ID of the employee to be deleted.
4. **Modifying Employee Details:**
   - Choose option 3 from the main menu to modify an employee's details.
   - Enter the ID of the employee to be modified.
   - Choose the field to modify (name, salary, phone number, etc.) and provide the new value.
5. **Searching for an Employee:**
   - Choose option 4 from the main menu to search for an employee.
   - Enter the employee's name for the search.
6. **Printing Employee Records:**
   - Choose option 5 from the main menu to print employee records.
   - Choose the sorting option ('N' for name, 'D' for date of birth, 'S' for salary).
   - Employee records will be displayed on the screen.
7. **Saving Employees to File:**
   - Choose option 6 from the main menu to save employee records to a file.
   - Enter the filename to save the data. The program will overwrite the existing file or create a new one if it doesn't exist.
8. **Quitting the Program:**
   - Choose option 7 from the main menu to quit the program.
   - The program will prompt for confirmation before quitting.

Main Algorithms Used

Search Algorithm

- The program uses a simple linear search algorithm to find an employee based on their name.

Sort Algorithms

1. **Sort By Name (sortByName function):**
    - Uses the Bubble Sort algorithm to sort the array of employees alphabetically by name.
2. **Sort By Salary (SortBySalary function):**
    - Uses the Bubble Sort algorithm to sort the array of employees in descending order of salary.
3. **Sort By Date of Birth (SortByDOB function):**
    - Uses the Bubble Sort algorithm to sort the array of employees in ascending order of date of birth.

These sorting algorithms provide a basic way to organize and display employee records based on different criteria.

Conclusion

The Employee Management System offers essential functionalities for managing employee data. Users can easily add, delete, modify, search, print, and save employee records. The system employs basic search and sorting algorithms to enhance the user experience. The code is designed to be user-friendly and straightforward, making it suitable for small-scale employee management tasks.

Code:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <unistd.h>

// Global variables for time-related operations

time_t t;
struct tm *local;

// Maximum number of employees the system can handle

#define MAX_EMPLOYEES 400

// ANSI color codes for terminal text formatting

#define RED "\x1b[38;5;203m"
#define GREEN "\x1b[38;5;42m"
#define YELLOW "\x1b[1;33m"
#define RESET "\x1b[0m"

typedef struct
{
    int day;
    int month;
    int year;
} date;

typedef struct
{
    char ID[8];
    char name[100];
    char salary[1000];
    date birth;
    char address[200];
    char phone[20];
    date En_date;
    char email[100];
} st_Employee;

// Array to store employee data
st_Employee Data_B_Employee[MAX_EMPLOYEES];

// Variable to keep track of the number of employees
int Num_of_Employee = 0;

// Function prototypes
```

```c
void greet_based_on_time();
void load_from_file(const char *filename);
void clear_buffer();
void queryEmployee(char *s);
void addEmployee();
void validateID(char id[], int size);
void FullNameValidation(char fullname[100], int size);
void SalaryValidation(char num[]);
void DateValidation(int *D, int *M, int *Y);
void AddressValidation(char address[], int size);
void PhoneNumberValidation(char phonenumber[100], int size);
int EmailValidation(char *email);
void getEmailAndValidate(char email[], int size);
void deleteEmployee(char *W_ID);
void modifyEmployee(char W_ID[]);
void printEmployees(char sortBy);
void sortByName();
void SortBySalary();
void SortByDOB();
void saveToFile(const char *fileName);
int confirmQuit();
int main()
{
    greet_based_on_time();

    char filename[100];
    char id_delete[10];
    char S_ID[10];
    char s[100];
    char sortBy;
    int choice;

    printf("\nEnter the Filename to load from [if exists]: ");
    int flag = 0;
    do
    {
        if (scanf("%99s", filename) != 1)
        {
            printf(RED "Error reading file name\n" RESET);
            clear_buffer();
            flag = 1;
        }
        else
        {
            flag = 0;
        }
    } while (flag);
    load_from_file(filename);

    printf(GREEN "\nLoaded %d Employees.\n" RESET, Num_of_Employee);

    do
    {
```

```
printf("\nMenu:\n");
printf("1. Add Employee\n");
printf("2. Delete Employee\n");
printf("3. Modify Employee\n");
printf("4. Search Employee\n");
printf("5. Print Employees\n");
printf("6. Save Employees to File\n");
printf("7. Quit\n");
printf("Enter your choice (1-7): ");

if (scanf("%d", &choice) != 1)
{
    clear_buffer();
    printf(RED "Invalid input...please enter an integer value.\n" RESET);
    continue;
}

switch (choice)
{
case 1:
    printf(GREEN "\nSelected: Add Employee\n" RESET);
    addEmployee();
    break;
case 2:

    printf(GREEN "\nSelected: Delete Employee\n" RESET);

    validateID(id_delete, sizeof(id_delete));
    deleteEmployee(id_delete);
    break;
case 3:
    printf(GREEN "\nSelected: Modify Employee\n" RESET);

    validateID(S_ID, sizeof(S_ID));
    modifyEmployee(S_ID);
    break;
case 4:

    printf(GREEN "\nSelected: Search Employee\n" RESET);
    printf("Enter the employee's name: ");
    fgets(s, sizeof(s), stdin);
    queryEmployee(s);
    break;
case 5:

    printf(GREEN "Selected: Print Employees\n" RESET);
    printf("Sort by (N)ame, (D)ate of Birth, or (S)alary: ");
    if (scanf(" %c", &sortBy) != 1)
    {
        clear_buffer();
        printf(RED "Invalid input...please enter a character value['N','D','S'].\n" RESET);
    }
    printEmployees(sortBy);
```

```c
            break;
        case 6:

            printf(GREEN "Selected: Save Employees to File\n");
            saveToFile(filename);
            break;
        case 7:
            if (confirmQuit())
            {
                exit(10);
            }
            break;

        default:
            printf(RED "\nInvalid input...try again.\n" RESET);
            break;
        }
    } while (choice != 7);

    return 0;
}

void greet_based_on_time()
{
    time_t t;
    struct tm *currentTime;

    // Get the current time
    time(&t);
    currentTime = localtime(&t);

    // Extract the hour from the current time
    int hour = currentTime->tm_hour;

    // Print greetings based on the current time
    if (hour >= 0 && hour < 12)
    {
        printf(GREEN "Good Morning! \n" RESET);
    }
    else if (hour >= 12 && hour < 18)
    {
        printf(GREEN "Good Afternoon!\n" RESET);
    }
    else
    {
        printf(GREEN "Good Evening!\n" RESET);
    }
}

void load_from_file(const char *filename)
{
    FILE *originalFile = fopen(filename, "r");
    if (originalFile == NULL)
    {

        printf("%s does Not exist creating %s \n", filename, filename);
        originalFile = fopen(filename, "w");
        if (originalFile == NULL)
        {
            printf(RED "Error creating %s file" RESET, filename);
            exit(2);
        }
        fclose(originalFile);
        return;
    }

    while (fscanf(originalFile, "%[^,],", Data_B_Employee[Num_of_Employee].ID) == 1)
    {
        fscanf(originalFile, " %[^,],", Data_B_Employee[Num_of_Employee].name);
        fscanf(originalFile, " %[^,],", Data_B_Employee[Num_of_Employee].salary);
        fscanf(originalFile, "%d-", &Data_B_Employee[Num_of_Employee].birth.day);
        fscanf(originalFile, "%d-", &Data_B_Employee[Num_of_Employee].birth.month);
        fscanf(originalFile, "%d,", &Data_B_Employee[Num_of_Employee].birth.year);
        fscanf(originalFile, " %[^,],", Data_B_Employee[Num_of_Employee].address);
        fscanf(originalFile, " %[^,],", Data_B_Employee[Num_of_Employee].phone);
        fscanf(originalFile, "%d-", &Data_B_Employee[Num_of_Employee].En_date.day);
        fscanf(originalFile, "%d-", &Data_B_Employee[Num_of_Employee].En_date.month);
        fscanf(originalFile, "%d,", &Data_B_Employee[Num_of_Employee].En_date.year);
        fscanf(originalFile, " %[^\n]\n", Data_B_Employee[Num_of_Employee].email);
        ++Num_of_Employee;
    }

    fclose(originalFile);
}
void clear_buffer()
{
    int clear;
    do
    {
        clear = getchar();
    } while (clear != '\n' && clear != EOF);
}
void queryEmployee(char *s)
{
    int flag = 0;
    for (int i = 0; i < Num_of_Employee; i++)
    {

        if (strstr(Data_B_Employee[i].name, s) != NULL)
        {
            printf("%s, %s, %s,%d-%d-%d, %s, %s,%d-%d-%d, %s\n", Data_B_Employee[i].ID,
Data_B_Employee[i].name, Data_B_Employee[i].salary, Data_B_Employee[i].birth.day,
Data_B_Employee[i].birth.month, Data_B_Employee[i].birth.year, Data_B_Employee[i].address,
Data_B_Employee[i].phone, Data_B_Employee[i].En_date.day, Data_B_Employee[i].En_date.month,
Data_B_Employee[i].En_date.year, Data_B_Employee[i].email);
            flag = 1;
        }
    }
    if (flag == 0)
    {
        printf(RED "\n Employee not found" RESET);
    }
}
void addEmployee()
{
    int n;
    int i = Num_of_Employee;
    t = time(NULL);
    local = localtime(&t);
    Data_B_Employee[i].En_date.year = local->tm_year + 1900;
    Data_B_Employee[i].En_date.month = local->tm_mon + 1;
    Data_B_Employee[i].En_date.day = local->tm_mday;
    printf("\nEnter the number of Employees to be added: ");

    if (scanf("%d", &n) != 1)
    {
        printf(RED "\nPlease Enter a integer value.\n" RESET);
        clear_buffer();
    }

    for (i = Num_of_Employee; i < (n + Num_of_Employee); i++)
    {
        printf("\nEmployee #%d: \n\n", i + 1);

        validateID(Data_B_Employee[i].ID, sizeof(Data_B_Employee[i].ID));

        FullNameValidation(Data_B_Employee[i].name, sizeof(Data_B_Employee[i].name));

        SalaryValidation(Data_B_Employee[i].salary);

        DateValidation(&Data_B_Employee[i].birth.day, &Data_B_Employee[i].birth.month,
&Data_B_Employee[i].birth.year);

        AddressValidation(Data_B_Employee[i].address, sizeof(Data_B_Employee[i].address));

        PhoneNumberValidation(Data_B_Employee[i].phone, sizeof(Data_B_Employee[i]));

        getEmailAndValidate(Data_B_Employee[i].email, sizeof(Data_B_Employee[i].email));
    }

    Num_of_Employee = Num_of_Employee + n;
}
void validateID(char id[], int size)
{
    char duplicate[10];

    while (1)
    {
        printf("Enter the ID of the employee: ");
```

```c
        clear_buffer();
        fgets(id, size, stdin);
        id[strcspn(id, "\n")] = '\0'; // Remove the newline character

        if (strcmp(id, "0") == 0)
        {
            printf("ID cannot equal 0. Re-enter the ID.\n");
        }
        else if (strlen(id) > 8)
        {
            printf("ID must not exceed 7 digits. Validation failed.\n");
        }
        else
        {
            int flag = 1;
            for (int i = 0; id[i] != '\0'; i++)
            {
                if (!isdigit(id[i]) || isspace(id[i]))
                {
                    printf("Please enter a valid ID.\n");
                    flag = 0;
                    break;
                }
            }

            if (flag == 1)
            {
                if (strcmp(id, duplicate) == 0)
                {
                    printf("ID already exists. Re-enter the ID.\n");
                }
                else
                {
                    strcpy(duplicate, id);
                    break;
                }
            }
        }
    }
}
void FullNameValidation(char fullname[100], int size)
{
    int flag = 0;

    while (flag == 0)
    {
        printf("Enter the name:");

        fgets(fullname, size, stdin);
        fullname[strcspn(fullname, "\n")] = '\0'; // function for removing \n
        flag = 1;
        for (int i = 0; fullname[i] != '\0'; i++)
        {
            if (!(isalpha(fullname[i]) || isspace(fullname[i])))
            {
                flag = 0;
                printf(RED "Please enter a valid full name.\n" RESET);
                break;
            }
        }
    }
}
void SalaryValidation(char num[])
{
    int i;
    int flag = 0;
    while (flag == 0)
    {
        printf("Enter employee's salary : ");
        scanf("%s", num);
        flag = 1;
        for (i = 0; num[i] != '\0'; i++)
        {
            if (strcmp(num, "0") == 0)
            {
                printf(RED "Please enter a valid salary.\n" RESET);
                flag = 0;
                break;
            }
            else if (!isdigit(num[i]) || isalpha(num[i]) || num[0] == 0)
            {
                printf(RED "Please enter a valid salary.\n" RESET);
                flag = 0;
                break;
            }
            else
            {
                flag = 1;
            }
        }
    }
}
void DateValidation(int *D, int *M, int *Y)
{
    int flag = 0;
    while (flag == 0)
    {
        printf("Enter Date of birth (DD-MM-YYYY): ");
        if (scanf("%d-%d-%d", D, M, Y) != 3)
        {
            clear_buffer();
            printf(RED "Invalid input! Please enter numeric values.\n" RESET);
        }
        if ((*D > 31 || *D <= 0) || (*M > 12 || *M <= 0) || (*Y < 1963 || *Y >= 2023))
        {
            printf(RED "Invalid date! Please re-enter.\n" RESET);
        }
        else if (*D > 30 && (*M == 4 || *M == 6 || *M == 9 || *M == 11))
        {
            printf(RED "Invalid date! Please re-enter.\n" RESET);
        }
        else if (*D > 28 && *M == 2)
        {
            printf(RED "Invalid date! Please re-enter.\n" RESET);
        }
        else
        {
            flag = 1;
        }
    }
}
void AddressValidation(char address[], int size)
{

    int i, flag = 0;

    while (flag == 0)
    {
        printf("Enter the employee's address:");
        clear_buffer();
        fgets(address, size, stdin);
        address[strcspn(address, "\n")] = '\0';

        for (i = 0; address[i]; i++)
        {
            if (isalpha(address[i]))
            {
                if (isalpha(address[i]) || isdigit(address[i]))
                {
                    flag = 1;
                    break;
                }
                else if (!isalpha(address[i]) || isdigit(address[i]))
                {
                    flag = 0;
                    break;
                }
            }
        }
    }
}
void PhoneNumberValidation(char phonenumber[100], int size)
{

    int i, flag = 0;

    while (flag == 0)
    {
        printf("Enter the employee's phone number: ");
```

```c
        fgets(phonenumber, size, stdin);
        phonenumber[strcspn(phonenumber, "\n")] = '\0'; // function for removing \n
        flag = 1;
        if (strlen(phonenumber) != 11)
        {
            flag = 0;
        }
        else
        {
            for (i = 0; phonenumber[i] != '\0'; i++)
            {
                if (!isdigit(phonenumber[i]))
                {
                    flag = 0;
                    break;
                }
            }
        }
        if (flag == 0)
        {
            printf(RED "Please enter a valid 11-digit phone number.\n" RESET);
        }
    }
}
int EmailValidation(char *email)
{
    int atCount = 0;
    unsigned long i;

    for (i = 0; i < strlen(email); i++)
    {
        if (email[i] == '@')
        {
            atCount++;
        }

        if (email[i] == ' ' || email[i] == '/' || email[i] == ':' || email[i] == ';' || email[i] ==
'<' || email[i] == '>' || email[i] == ',' || email[i] == '[' || email[i] == ']')
        {
            return 0;
        }
    }

    if (atCount == 1)
    {
        if (email[0] != '@')
        {
            char *dot = strchr(email, '.');

            if (dot != NULL && dot > strchr(email, '@'))
            {
                return 1;
            }
        }
    }
    return 0;
}
void getEmailAndValidate(char email[], int size)
{
    do
    {
        printf("Enter employee's email address: ");
        fgets(email, size, stdin);
        email[strcspn(email, "\n")] = '\0'; // function for removing \n

        int statusOfValidation = EmailValidation(email);
        if (statusOfValidation)
        {

            break;
        }
        else
        {
            printf("Invalid email address. Please try again.\n");
        }
    } while (1);
}

void deleteEmployee(char *W_ID)
{
    int found = 0;

    for (int i = 0; i < Num_of_Employee; i++)
    {
        if (strcmp(Data_B_Employee[i].ID, W_ID) == 0)
        {
            found = 1;

            for (int j = i; j < Num_of_Employee - 1; j++)
            {
                Data_B_Employee[j] = Data_B_Employee[j + 1];
            }

            memset(&Data_B_Employee[Num_of_Employee - 1], 0, sizeof(st_Employee));

            Num_of_Employee--;
            break;
        }
    }

    if (found)
    {
        printf(GREEN "Employee with ID %s deleted successfully.\n" RESET, W_ID);
    }
    else
    {
        printf(RED "Employee with ID %s not found.\n" RESET, W_ID);
    }
}

void modifyEmployee(char W_ID[])
{
    int found = 0;
    int choice;

    for (int i = 0; i < Num_of_Employee; i++)
    {
        if (strcmp(Data_B_Employee[i].ID, W_ID) == 0)
        {
            found = 1;
            sleep(1);
            printf("Current details for Employee with ID %s:\n", W_ID);
            printf("Name: %s\n", Data_B_Employee[i].name);
            printf("Salary: %s\n", Data_B_Employee[i].salary);
            printf("Birth Date: %d-%d-%d\n", Data_B_Employee[i].birth.day,
Data_B_Employee[i].birth.month, Data_B_Employee[i].birth.year);
            printf("En_date Date: %d-%d-%d\n", Data_B_Employee[i].En_date.day,
Data_B_Employee[i].En_date.month, Data_B_Employee[i].En_date.year);
            printf("Address: %s\n", Data_B_Employee[i].address);
            printf("Phone: %s\n", Data_B_Employee[i].phone);
            printf("Email: %s\n\n", Data_B_Employee[i].email);

            do
            {
                sleep(1);
                printf("\n\nSelect field to modify:\n");
                printf("1. Full Name\n");
                printf("2. Salary\n");
                printf("3. Mobile Number\n");
                printf("4. Address\n");
                printf("5. Email\n");
                printf("6. Back to main menu\n");
                printf("Enter your choice (1-6): ");
                if (scanf("%d", &choice) != 1)
                {
                    clear_buffer();
                    printf(RED "Enter an integer value please..." RESET);
                    continue;
                }
                switch (choice)
                {
                case 1:
                    printf("Enter new full name: ");
                    FullNameValidation(Data_B_Employee[i].name, sizeof(Data_B_Employee[i].name));
                    break;
                case 2:
                    printf("Enter new salary: ");
                    SalaryValidation(Data_B_Employee[i].salary);
                    break;
                case 3:
                    PhoneNumberValidation(Data_B_Employee[i].phone, 
sizeof(Data_B_Employee[i].phone));
                    break;
                case 4:
                    printf("Enter new address:");

                    AddressValidation(Data_B_Employee[i].address, 
sizeof(Data_B_Employee[i].address));
                    break;
                case 5:

                    clear_buffer();
                    getEmailAndValidate(Data_B_Employee[i].email, 
sizeof(Data_B_Employee[i].email));
                    break;
                case 6:

                    return;
                default:
                    printf("Invalid choice. No modifications made.\n");
                    break;
                }
            } while (choice != 6);

            break;
        }
    }
    if (!found)
    {
        printf(RED "Employee with ID %s not found.\n" RESET, W_ID);
    }
}
void printEmployees(char sortBy)
{
    // Convert sortBy to uppercase
    sortBy = toupper((unsigned char)sortBy);

    if (Num_of_Employee == 0)
    {
        printf("No employees to print.\n");
        return;
    }

    switch (sortBy)
    {
    case 'N':
        sortByName();
        break;
    case 'D':

        SortByDOB();
        break;
```

```c
    case 'S':

        SortBySalary();
        break;
    default:
        printf("Invalid sorting option.\n");
        return;
    }
    printf("\n");
    sleep(1);
    for (int i = 0; i < Num_of_Employee; i++)
    {
        printf("%s, %s, %s,%d-%d-%d, %s, %s,%d-%d-%d, %s\n", Data_B_Employee[i].ID,
               Data_B_Employee[i].name, Data_B_Employee[i].salary, Data_B_Employee[i].birth.day,
               Data_B_Employee[i].birth.month, Data_B_Employee[i].birth.year,
Data_B_Employee[i].address,
               Data_B_Employee[i].phone, Data_B_Employee[i].En_date.day,
Data_B_Employee[i].En_date.month,
               Data_B_Employee[i].En_date.year, Data_B_Employee[i].email);
    }
    sleep(1);
}

void sortByName()
{
    st_Employee temp;
    for (int i = 0; i < Num_of_Employee - 1; i++)
    {
        for (int j = i + 1; j < Num_of_Employee; j++)
        {
            if (strcmp(Data_B_Employee[i].name, Data_B_Employee[j].name) > 0)
            {
                temp = Data_B_Employee[i];
                Data_B_Employee[i] = Data_B_Employee[j];
                Data_B_Employee[j] = temp;
            }
        }
    }
}

void SortBySalary()
{
    st_Employee temp;
    for (int i = 0; i < Num_of_Employee - 1; i++)
    {
        for (int j = i + 1; j < Num_of_Employee; j++)
        {
            if (strcmp(Data_B_Employee[i].salary, Data_B_Employee[j].salary) < 0)
            {
                temp = Data_B_Employee[i];
                Data_B_Employee[i] = Data_B_Employee[j];
                Data_B_Employee[j] = temp;
            }
        }
    }
}

void SortByDOB()
{
    st_Employee temp;
    for (int i = 0; i < Num_of_Employee - 1; i++)
    {
        for (int j = i + 1; j < Num_of_Employee; j++)
        {
            if (Data_B_Employee[i].birth.year > Data_B_Employee[j].birth.year)
            {
                temp = Data_B_Employee[i];
                Data_B_Employee[i] = Data_B_Employee[j];
                Data_B_Employee[j] = temp;
            }
            else if (Data_B_Employee[i].birth.year == Data_B_Employee[j].birth.year)
            {
                if (Data_B_Employee[i].birth.month > Data_B_Employee[j].birth.month)
                {
                    temp = Data_B_Employee[i];
                    Data_B_Employee[i] = Data_B_Employee[j];
                    Data_B_Employee[j] = temp;
                }
                else if (Data_B_Employee[i].birth.month == Data_B_Employee[j].birth.month)
                {
                    if (Data_B_Employee[i].birth.day > Data_B_Employee[j].birth.day)
                    {
                        temp = Data_B_Employee[i];
                        Data_B_Employee[i] = Data_B_Employee[j];
                        Data_B_Employee[j] = temp;
                    }
                }
            }
        }
    }
}

void saveToFile(const char *filename)
{

    FILE *save_File = fopen(filename, "w");

    if (save_File == NULL)
    {
        printf(RED "Error opening file" RESET);
        exit(4);
    }

    for (int i = 0; i < Num_of_Employee; i++)
    {
        if (strcmp(Data_B_Employee[i].ID, "NULL") == 0)
        {
            continue;
        }
        else
        {
            fprintf(save_File, "%s, %s, %s,%d-%d-%d, %s, %s,%d-%d-%d, %s", Data_B_Employee[i].ID,
                    Data_B_Employee[i].name, Data_B_Employee[i].salary,
Data_B_Employee[i].birth.day,
                    Data_B_Employee[i].birth.month, Data_B_Employee[i].birth.year,
Data_B_Employee[i].address,
                    Data_B_Employee[i].phone, Data_B_Employee[i].En_date.day,
Data_B_Employee[i].En_date.month,
                    Data_B_Employee[i].En_date.year, Data_B_Employee[i].email);

            if (i < Num_of_Employee - 1)
            {
                fprintf(save_File, "\n");
            }
        }
    }
    fclose(save_File);
    printf(GREEN "\nSuccessfully Saved to file.\n" RESET);
}
int confirmQuit()
{
    char choice;

    printf(YELLOW "!! Make sure to save your progress. !!\n");
    printf("Are you sure you want to quit?\n" RESET);
    printf("Enter (Y) Yes, (N) No: ");
    clear_buffer();

    if (scanf(" %c", &choice) != 1)
    {
        clear_buffer();
        printf(RED "\nPlease enter 'Y' or 'N' only\n" RESET);
        return 0; // Retry the quit confirmation
    }

    if (choice == 'Y' || choice == 'y')
    {
        printf(GREEN "Quitting the program...\n" RESET);
        return 1; // Quit the program
    }
    else if (choice == 'N' || choice == 'n')
    {
        return 0; // Continue with the program
    }
    else
    {
        printf(RED "\nInvalid input...Back to the main menu.\n" RESET);
        return 0; // Retry the quit confirmation    }}
```