**Department of Computer science and Engineering**

**CS 204:Design and Analysis of Algorithm**

**Project Title:OBE Implementation**

***Team Deatail****s:*

**Team Name :** The White Walkers

**Team project:** Program level objective Setting

**Team leader:** Srinadh– AP23110011147

**Team members:**

Srinadh Reddy- AP23110011147

Ratnesh G - AP23110011134

Syam Kolli - AP23110010711

Vivek ch - AP23110011135

Yogesh P - AP23110011100

***SUBMITTED TO***

**Designed by**

**GAVASKAR S,**

**Assistant Professor(Ad),**

**Department of CSE,**

**SRM University - AP.**

**INDEX**

**Introduction**

**Modules in the project:**

**Architecture Diagram**

Instructions:

**Module Description**

Programming Details naming conventions to be used:

Field/table details : Program level objective settings

Algorithm Details:

(i)Sorting

(ii)Searching

(ii) Storing the details in a text file 6

**Sample coding Template**

OBE\_MAIN\_The\_White\_Walkers.c

Instruction to use Sample Code:

**ChatGPT Usage**

Program Generated by ChatGPT

Instruction to use ChatGPT or other LLM Models

**Introduction**

This project module manages records for various "programs," where each program is defined by its unique ID, code, name, and description. The application supports basic CRUD (Create, Retrieve, Update, Delete) operations on these records, with functionalities for sorting and searching. The data is stored in a text file for persistence, ensuring that program records remain accessible across sessions. This project demonstrates effective data management techniques, basic algorithm comparisons, and efficiency evaluation for sorting and searching operations.

**Project Module:**

**Program Level Objective Setting**

The primary module in this project is Program Management, focused on handling records for different programs.

This module allows:

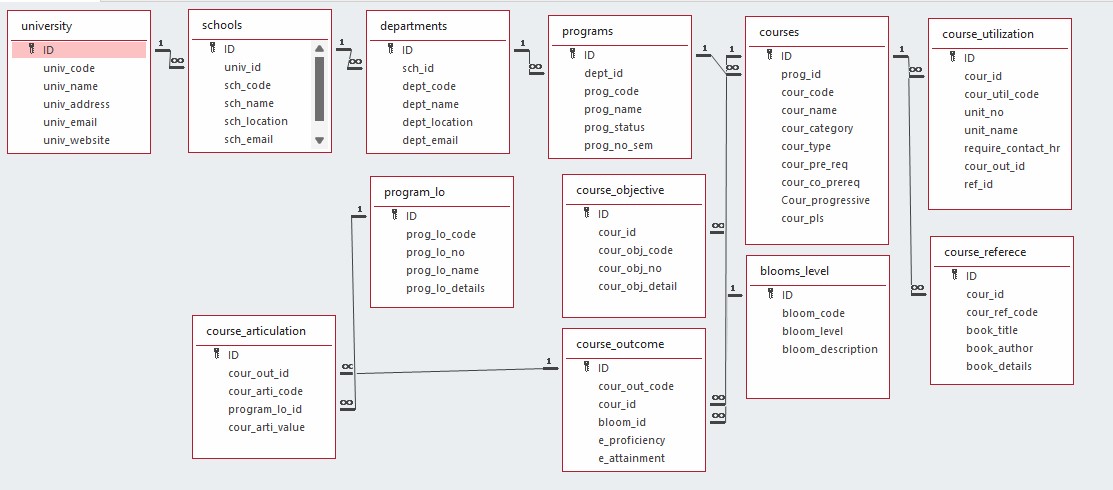
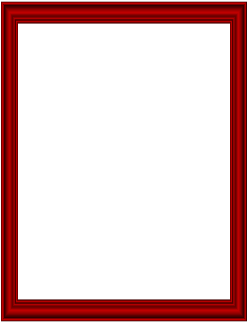
**ID-based identification**: Each program has a unique integer ID.

**Attribute-based searching and sorting**: Programs can be located and organized by their code, name for easier access and management.

**CRUD Operations**: Users can create, retrieve, update, and delete program records efficiently.

The Program Management module leverages Selection Sort for sorting and Linear Search for basic searching, providing a straightforward approach to organizing and locating records.

**Architecture Diagram**



# 

# **Module Description**

**Module Name:** *Program Management*

**Module Description:**

The Program Management module facilitates CRUD operations on program records. Users can:

**Add a new program record.**

**View (retrieve) existing records.**

**Update specific program details.**

**Delete program records.**

Data is stored in a file (programs\_setting.txt), and each change (create, update, delete) is reflected in this file. Sorting and searching capabilities are also provided, allowing users to organize and locate program records based on attributes like program code and name.

Programming Details naming conventions to be used:

* **File name:** ● **Function/method name**

○ **Create:** The\_White\_Walkers\_create\_program

○ **Update:** The\_White\_Walkers\_update\_program

○ **Retrieve:** The\_White\_Walkers\_retrieve\_programs

○ **Delete:** The\_White\_Walkers\_delete\_program

○**Sorting:** The\_White\_Walkers\_sort\_by\_field

○**Searching:** The\_White\_Walkers\_search\_by\_field

○ **Comparison(both Searching & Sorting)**:

■ For Searching - The\_White\_Walkers\_compare\_searching\_algorithms

■ For Sorting - The\_White\_Walkers\_compare\_sorting\_algorithms

○ **Time Complexity(both searching and Sorting):**

■ For Searching/Sorting- The\_White\_Walkers \_display\_time\_complexity

○ **Algorithm Details(pseudocode or steps)(both searching and Sorting):**

■ For Searching: The\_White\_Walkers\_display\_pseudocode

●**File name(for storing the details):**

○ File name to be used is:- Programs\_setting .txt

Field/table details: For Programs

|  |  |
| --- | --- |
| **Field Name** | **Data type** |
| Id | integer |
| program\_code | String |
| program\_name | String |
| Program\_desc | String |

Algorithm Details:

(i)Sorting

Sorting is based on attributes such as program\_code and program\_name. The module uses Selection Sort as its primary sorting algorithm and compares it with Bubble Sort:

**Primary Sorting Algorithm (Selection Sort):** This algorithm iteratively finds the minimum element and places it at the beginning. While simple and effective for smaller datasets, its complexity makes it slower on larger datasets.

**Comparison Algorithm (Bubble Sort):** Similar in complexity to Selection Sort, Bubble Sort is less efficient as it requires repeated swapping. However, comparing these algorithms highlights the limitations of basic sorting methods for larger datasets.

(ii)Searching

Searching enables users to find specific program records based on fields like program\_code and program\_name. Two algorithms are employed:

**Primary Searching Algorithm**: Linear Search is straightforward and works well with smaller datasets.

**Comparison Algorithm**: Binary Search (if the data is sorted), which is more efficient, with a time complexity of O(logn)

Each algorithm’s time complexity is presented, giving insight into performance differences between linear and binary search.

(ii) Storing the details in a text file

The details are stored in programs\_setting.txt and updated with each CRUD operation:

**Create**: Adds a new program entry.

**Update**: Modifies an existing entry.

**Delete**: Removes a specific program record.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

#define FILE\_NAME "programs\_setting.txt"

// Color codes for terminal output

#define COLOR\_RESET "\033[0m"

#define COLOR\_SUCCESS "\033[32m" // Green

#define COLOR\_ERROR "\033[31m"   // Red

#define COLOR\_INFO "\033[34m"    // Blue

// Program structure

typedef struct {

    int program\_id;

    char program\_code[20];

    char program\_name[50];

    char program\_desc[100];

} Program;

// Global array to store program data and a counter

Program programs[MAX];

int program\_count = 0;

// Function declarations

void The\_White\_Walkers\_load\_from\_file();

void The\_White\_Walkers\_store\_to\_file();

void The\_White\_Walkers\_create\_program();

void The\_White\_Walkers\_update\_program();

void The\_White\_Walkers\_retrieve\_programs();

void The\_White\_Walkers\_delete\_program();

void The\_White\_Walkers\_search\_by\_field();

void The\_White\_Walkers\_sort\_by\_field();

void The\_White\_Walkers\_compare\_sorting\_algorithms();

void The\_White\_Walkers\_compare\_searching\_algorithms();

void The\_White\_Walkers\_display\_time\_complexity();

void The\_White\_Walkers\_display\_pseudocode();

void The\_White\_Walkers\_display\_menu();

void The\_White\_Walkers\_display\_help();

void The\_White\_Walkers\_manual\_store\_to\_file(); // New function for manual storage

int The\_White\_Walkers\_is\_unique\_id(int id);

void The\_White\_Walkers\_bubble\_sort(const char \*field);

void The\_White\_Walkers\_selection\_sort(const char \*field);

int The\_White\_Walkers\_linear\_search(int id);

int The\_White\_Walkers\_binary\_search(int id);

void The\_White\_Walkers\_display\_table();

// Function to clear the input buffer

void The\_White\_Walkers\_clear\_input\_buffer() {

    int c;

    while ((c = getchar()) != '\n' && c != EOF);

}

// Main function

int main() {

    The\_White\_Walkers\_load\_from\_file();

    The\_White\_Walkers\_display\_menu();

    return 0;

}

// Function to display all programs in a tabular format

void The\_White\_Walkers\_display\_table() {

    if (program\_count == 0) {

        printf(COLOR\_ERROR "No programs to display.\n" COLOR\_RESET);

        return;

    }

    // Print table header

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

    printf("| No. | Program ID | Program Code       | Program Name                 | Description                            |\n");

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

    // Print each program entry in a row

    for (int i = 0; i < program\_count; i++) {

        printf("| %-3d | %-10d | %-18s | %-28s | %-38s |\n",

               i + 1,

               programs[i].program\_id,

               programs[i].program\_code,

               programs[i].program\_name,

               programs[i].program\_desc);

    }

    // Print table footer

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

}

// Function to load data from file with error handling

void The\_White\_Walkers\_load\_from\_file() {

    FILE \*file = fopen(FILE\_NAME, "r");

    if (file == NULL) {

        perror("Error opening file. Starting with an empty record.");

        return;

    }

    program\_count = 0;

    while (fscanf(file, "%d %19s %49s %[^\n]",

                  &programs[program\_count].program\_id,

                  programs[program\_count].program\_code,

                  programs[program\_count].program\_name,

                  programs[program\_count].program\_desc) != EOF) {

        program\_count++;

    }

    fclose(file);

    printf(COLOR\_SUCCESS "Data loaded successfully from file.\n" COLOR\_RESET);

    The\_White\_Walkers\_display\_table();

}

// Function to store data to file with error handling

void The\_White\_Walkers\_store\_to\_file() {

    FILE \*file = fopen(FILE\_NAME, "w");

    if (file == NULL) {

        perror("Error opening file for saving");

        return;

    }

    for (int i = 0; i < program\_count; i++) {

        fprintf(file, "%d %s %s %s\n", programs[i].program\_id, programs[i].program\_code,

                programs[i].program\_name, programs[i].program\_desc);

    }

    fclose(file);

    printf(COLOR\_SUCCESS "Data saved to file.\n" COLOR\_RESET);

}

// Function for manual storage of data to file

void The\_White\_Walkers\_manual\_store\_to\_file() {

    The\_White\_Walkers\_store\_to\_file(); // Call the existing store\_to\_file function

}

// Check if the Program ID is unique

int The\_White\_Walkers\_is\_unique\_id(int id) {

    for (int i = 0; i < program\_count; i++) {

        if (programs[i].program\_id == id) {

            return 0; // ID is not unique

        }

    }

    return 1; // ID is unique

}

// Function to create a new program record with error handling

void The\_White\_Walkers\_create\_program() {

    if (program\_count >= MAX) {

        printf(COLOR\_ERROR "Program list is full!\n" COLOR\_RESET);

        return;

    }

    Program p;

    int id\_valid = 0;

    // Read Program ID with validation

    while (!id\_valid) {

        printf("Enter Program ID (integer): ");

        if (scanf("%d", &p.program\_id) == 1 && The\_White\_Walkers\_is\_unique\_id(p.program\_id)) {

            id\_valid = 1;

        } else {

            printf(COLOR\_ERROR "Invalid input or ID already exists. Please enter a unique integer.\n" COLOR\_RESET);

            The\_White\_Walkers\_clear\_input\_buffer();

        }

    }

    The\_White\_Walkers\_clear\_input\_buffer();

    // Read Program Code

    printf("Enter Program Code: ");

    if (scanf("%19s", p.program\_code) != 1) {

        printf(COLOR\_ERROR "Error reading Program Code. Please try again.\n" COLOR\_RESET);

        return;

    }

    The\_White\_Walkers\_clear\_input\_buffer();

    // Read Program Name

    printf("Enter Program Name: ");

    if (scanf("%49s", p.program\_name) != 1) {

        printf(COLOR\_ERROR "Error reading Program Name. Please try again.\n" COLOR\_RESET);

        return;

    }

    The\_White\_Walkers\_clear\_input\_buffer();

    // Read Program Description

    printf("Enter Program Description: ");

    if (scanf(" %99[^\n]", p.program\_desc) != 1) {

        printf(COLOR\_ERROR "Error reading Program Description. Please try again.\n" COLOR\_RESET);

        return;

    }

    programs[program\_count++] = p;

    printf(COLOR\_SUCCESS "Program created successfully!\n" COLOR\_RESET);

}

// Update a program record with error handling

void The\_White\_Walkers\_update\_program() {

    int id;

    int found = 0;

    printf("Enter Program ID to update: ");

    if (scanf("%d", &id) != 1) {

        printf(COLOR\_ERROR "Invalid input. Please enter an integer.\n" COLOR\_RESET);

        The\_White\_Walkers\_clear\_input\_buffer();

        return;

    }

    The\_White\_Walkers\_clear\_input\_buffer();

    for (int i = 0; i < program\_count; i++) {

        if (programs[i].program\_id == id) {

            found = 1;

            printf("Enter new Program Code: ");

            scanf("%19s", programs[i].program\_code);

            The\_White\_Walkers\_clear\_input\_buffer();

            printf("Enter new Program Name: ");

            scanf("%49s", programs[i].program\_name);

            The\_White\_Walkers\_clear\_input\_buffer();

            printf("Enter new Program Description: ");

            scanf(" %99[^\n]", programs[i].program\_desc);

            printf(COLOR\_SUCCESS "Program updated successfully!\n" COLOR\_RESET);

            break;

        }

    }

    if (!found) {

        printf(COLOR\_ERROR "Program with ID %d not found.\n" COLOR\_RESET, id);

    }

}

// Retrieve and display all program records with validation

void The\_White\_Walkers\_retrieve\_programs() {

    if (program\_count == 0) {

        printf(COLOR\_ERROR "No programs to display.\n" COLOR\_RESET);

        return;

    }

    printf("\nList of Programs:\n");

    // for (int i = 0; i < program\_count; i++) {

    //     printf("ID: %d\nCode: %s\nName: %s\nDescription: %s\n\n",

    //            programs[i].program\_id, programs[i].program\_code,

    //            programs[i].program\_name, programs[i].program\_desc);

    // }

    The\_White\_Walkers\_display\_table();

}

// Delete a program record with error handling

void The\_White\_Walkers\_delete\_program() {

    int id;

    int found = 0;

    printf("Enter Program ID to delete: ");

    if (scanf("%d", &id) != 1) {

        printf(COLOR\_ERROR "Invalid input. Please enter an integer.\n" COLOR\_RESET);

        The\_White\_Walkers\_clear\_input\_buffer();

        return;

    }

    The\_White\_Walkers\_clear\_input\_buffer();

    for (int i = 0; i < program\_count; i++) {

        if (programs[i].program\_id == id) {

            found = 1;

            for (int j = i; j < program\_count - 1; j++) {

                programs[j] = programs[j + 1];

            }

            program\_count--;

            printf(COLOR\_SUCCESS "Program deleted successfully!\n" COLOR\_RESET);

            break;

        }

    }

    if (!found) {

        printf(COLOR\_ERROR "Program with ID %d not found.\n" COLOR\_RESET, id);

    }

}

// Helper function to convert a string to lowercase

void to\_lowercase(char \*str) {

    for (int i = 0; str[i]; i++) {

        str[i] = tolower((unsigned char)str[i]);

    }

}

// Search program by a specified field with substring matching (case-insensitive)

void The\_White\_Walkers\_search\_by\_field() {

    char input[50];

    int found = 0, choice;

    printf("Choose field to search by:\n1. Program Code\n2. Program Name\nEnter choice: ");

    scanf("%d", &choice);

    The\_White\_Walkers\_clear\_input\_buffer();

    printf("Enter part of the text to search: ");

    if (scanf("%49s", input) != 1) {

        printf(COLOR\_ERROR "Error reading input.\n" COLOR\_RESET);

        The\_White\_Walkers\_clear\_input\_buffer();

        return;

    }

    The\_White\_Walkers\_clear\_input\_buffer();

    to\_lowercase(input); // Convert input to lowercase

    for (int i = 0; i < program\_count; i++) {

        char field\_text[50];

        if (choice == 1) {

            strncpy(field\_text, programs[i].program\_code, sizeof(field\_text) - 1);

        } else if (choice == 2) {

            strncpy(field\_text, programs[i].program\_name, sizeof(field\_text) - 1);

        } else {

            printf(COLOR\_ERROR "Invalid choice.\n" COLOR\_RESET);

            return;

        }

        field\_text[sizeof(field\_text) - 1] = '\0'; // Ensure null termination

        to\_lowercase(field\_text); // Convert field text to lowercase

        if (strstr(field\_text, input) != NULL) { // Case-insensitive match

            printf("ID: %d\nCode: %s\nName: %s\nDescription: %s\n\n",

                   programs[i].program\_id, programs[i].program\_code,

                   programs[i].program\_name, programs[i].program\_desc);

            found = 1;

        }

    }

    if (!found) {

        printf(COLOR\_INFO "No matching program found.\n" COLOR\_RESET);

    }

}

// Sort programs by a specified field using a chosen algorithm

void The\_White\_Walkers\_sort\_by\_field() {

    char field[20];

    int algorithm\_choice;

    printf("Choose sorting algorithm:\n1. Bubble Sort\n2. Selection Sort\nEnter choice: ");

    scanf("%d", &algorithm\_choice);

    The\_White\_Walkers\_clear\_input\_buffer();

    printf("Enter field to sort by (code, name): ");

    scanf("%19s", field);

    The\_White\_Walkers\_clear\_input\_buffer();

    if (algorithm\_choice == 1) {

        The\_White\_Walkers\_bubble\_sort(field);

    } else if (algorithm\_choice == 2) {

        The\_White\_Walkers\_selection\_sort(field);

    } else {

        printf(COLOR\_ERROR "Invalid sorting algorithm choice.\n" COLOR\_RESET);

    }

}

// Bubble sort implementation for programs

void The\_White\_Walkers\_bubble\_sort(const char \*field) {

    for (int i = 0; i < program\_count - 1; i++) {

        for (int j = 0; j < program\_count - i - 1; j++) {

            int compare = 0;

            if (strcmp(field, "code") == 0) {

                compare = strcmp(programs[j].program\_code, programs[j + 1].program\_code);

            } else if (strcmp(field, "name") == 0) {

                compare = strcmp(programs[j].program\_name, programs[j + 1].program\_name);

            }

            if (compare > 0) {

                Program temp = programs[j];

                programs[j] = programs[j + 1];

                programs[j + 1] = temp;

            }

        }

    }

    printf(COLOR\_SUCCESS "Programs sorted by %s using Bubble Sort!\n" COLOR\_RESET, field);

    The\_White\_Walkers\_retrieve\_programs();

}

// Selection sort implementation for programs

void The\_White\_Walkers\_selection\_sort(const char \*field) {

    for (int i = 0; i < program\_count - 1; i++) {

        int min\_index = i;

        for (int j = i + 1; j < program\_count; j++) {

            int compare = 0;

            if (strcmp(field, "code") == 0) {

                compare = strcmp(programs[j].program\_code, programs[min\_index].program\_code);

            } else if (strcmp(field, "name") == 0) {

                compare = strcmp(programs[j].program\_name, programs[min\_index].program\_name);

            }

            if (compare < 0) {

                min\_index = j;

            }

        }

        Program temp = programs[min\_index];

        programs[min\_index] = programs[i];

        programs[i] = temp;

    }

    printf(COLOR\_SUCCESS "Programs sorted by %s using Selection Sort!\n" COLOR\_RESET, field);

    The\_White\_Walkers\_retrieve\_programs();

}

// Compare sorting algorithms

void The\_White\_Walkers\_compare\_sorting\_algorithms() {

    char field[20];

    printf("Enter field to sort by for comparison (code, name): ");

    scanf("%19s", field);

    The\_White\_Walkers\_clear\_input\_buffer();

    printf("\nSorting using Bubble Sort:\n");

    The\_White\_Walkers\_bubble\_sort(field);

    printf("\nSorting using Selection Sort:\n");

    The\_White\_Walkers\_selection\_sort(field);

}

// Linear search implementation

int The\_White\_Walkers\_linear\_search(int id) {

    for (int i = 0; i < program\_count; i++) {

        if (programs[i].program\_id == id) {

            return i;

        }

    }

    return -1; // Not found

}

// Binary search implementation

int The\_White\_Walkers\_binary\_search(int id) {

    int left = 0, right = program\_count - 1;

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (programs[mid].program\_id == id) {

            return mid; // Found

        }

        if (programs[mid].program\_id < id) {

            left = mid + 1; // Search right half

        } else {

            right = mid - 1; // Search left half

        }

    }

    return -1; // Not found

}

void The\_White\_Walkers\_compare\_searching\_algorithms() {

    int id\_to\_search;

    printf("Enter Program ID to search: ");

    scanf("%d", &id\_to\_search);

    The\_White\_Walkers\_clear\_input\_buffer();

    printf("Comparing searching algorithms...\n");

    // Linear search

    int linear\_index = The\_White\_Walkers\_linear\_search(id\_to\_search);

    if (linear\_index != -1) {

        printf("Linear Search found Program ID %d at index %d.\n", id\_to\_search, linear\_index);

    } else {

        printf("Linear Search did not find Program ID %d.\n", id\_to\_search);

    }

    // Binary search (assumes programs are sorted by ID)

    int binary\_index = The\_White\_Walkers\_binary\_search(id\_to\_search);

    if (binary\_index != -1) {

        printf("Binary Search found Program ID %d at index %d.\n", id\_to\_search, binary\_index);

    } else {

        printf("Binary Search did not find Program ID %d.\n", id\_to\_search);

    }

}

// Display time complexity for sorting and searching algorithms

void The\_White\_Walkers\_display\_time\_complexity() {

    printf("Time Complexity Analysis:\n");

    printf("Bubble Sort: O(n^2)\n");

    printf("Selection Sort: O(n^2)\n");

    printf("Linear Search: O(n)\n");

    printf("Binary Search: O(log n)\n");

}

// Display pseudocode for sorting and searching algorithms

void The\_White\_Walkers\_display\_pseudocode() {

    printf("Pseudocode for Bubble Sort:\n");

    printf("for i from 0 to n-1\n");

    printf("   for j from 0 to n-i-1\n");

    printf("      if programs[j] > programs[j+1]\n");

    printf("         swap programs[j] and programs[j+1]\n\n");

    printf("Pseudocode for Selection Sort:\n");

    printf("for i from 0 to n-1\n");

    printf("   min\_index = i\n");

    printf("   for j from i+1 to n\n");

    printf("      if programs[j] < programs[min\_index]\n");

    printf("         min\_index = j\n");

    printf("   swap programs[i] and programs[min\_index]\n\n");

    printf("Pseudocode for Linear Search:\n");

    printf("for each element in programs\n");

    printf("   if element matches search criteria\n");

    printf("      return element\n\n");

}

// Display help information

void The\_White\_Walkers\_display\_help() {

    printf("Help Menu:\n");

    printf("1. Create Program: Add a new program to the list.\n");

    printf("2. Update Program: Modify an existing program's details.\n");

    printf("3. Retrieve Programs: Display all programs in the list.\n");

    printf("4. Delete Program: Remove a program from the list.\n");

    printf("5. Search by Field: Find a program using its code or name.\n");

    printf("6. Sort by Field: Organize programs based on code or name.\n");

    printf("7. Compare Sorting Algorithms: Compare performance of sorting methods.\n");

    printf("8. Display Time Complexity: Show time complexities of various algorithms.\n");

    printf("9. Display Pseudocode: View pseudocode for algorithms used.\n");

    printf("10. Manually Save to File: Save program data to a file manually.\n");

    printf("11. Exit: Close the program.\n");

}

// Display menu and handle choices with validation

void The\_White\_Walkers\_display\_menu() {

    int choice;

    do {

        printf("\n1. Create Program\n2. Update Program\n3. Retrieve Programs\n4. Delete Program\n");

        printf("5. Search by Field\n6. Sort by Field\n7. Compare Sorting Algorithms\n");

        printf("8. Compare Searching Algorithms\n");

        printf("9. Display Time Complexity\n10. Display Pseudocode\n11. Store/Save to File\n12. Help\n13. Exit\n");

        printf("Enter your choice: ");

        if (scanf("%d", &choice) != 1) {

            printf(COLOR\_ERROR "Invalid input. Please enter a number between 1 and 12.\n" COLOR\_RESET);

            The\_White\_Walkers\_clear\_input\_buffer();

            continue;

        }

        switch (choice) {

            case 1: The\_White\_Walkers\_create\_program(); break;

            case 2: The\_White\_Walkers\_update\_program(); break;

            case 3: The\_White\_Walkers\_retrieve\_programs(); break;

            case 4: The\_White\_Walkers\_delete\_program(); break;

            case 5: The\_White\_Walkers\_search\_by\_field(); break;

            case 6: The\_White\_Walkers\_sort\_by\_field(); break;

            case 7: The\_White\_Walkers\_compare\_sorting\_algorithms(); break;

            case 8: The\_White\_Walkers\_compare\_searching\_algorithms(); break;

            case 9: The\_White\_Walkers\_display\_time\_complexity(); break;

            case 10: The\_White\_Walkers\_display\_pseudocode(); break;

            case 11: The\_White\_Walkers\_manual\_store\_to\_file(); break; // New case for manual saving

            case 12: The\_White\_Walkers\_display\_help(); break;

            case 13: printf(COLOR\_SUCCESS "Exiting program.\n" COLOR\_RESET); exit(0);

            default: printf(COLOR\_ERROR "Invalid choice! Please enter a number between 1 and 12.\n" COLOR\_RESET);

        }

    } while (choice != 12);

}

# **Comparison of Sorting Algorithms**

1. **Selection Sort (Primary Algorithm)**

**Advantages**: Simple to understand and implement, suitable for small datasets.

**Disadvantages:** Inefficient on larger datasets due to its O(n^2) time complexity

2**. Bubble Sort (Comparison Algorithm)**

**Advantages**: Simple to understand and implement.

**Disadvantages:** Inefficient for large datasets due to its O(n^2) time complexity.

**Time Complexity:**

**Selection Sort**: O(n^2)

**Bubble Sort**:O(n^2)

# **Comparison of Searching Algorithms**

**1. Linear Search (Primary Algorithm**)

**Advantages**: Works well for unsorted data, straightforward to implement.

**Disadvantages:** Inefficient for large datasets as it checks each element (time complexity O(n)).

**2. Binary Search (Comparison Algorithm)**

**Advantages**: Efficient with a time complexity of O(logn) but requires sorted data.

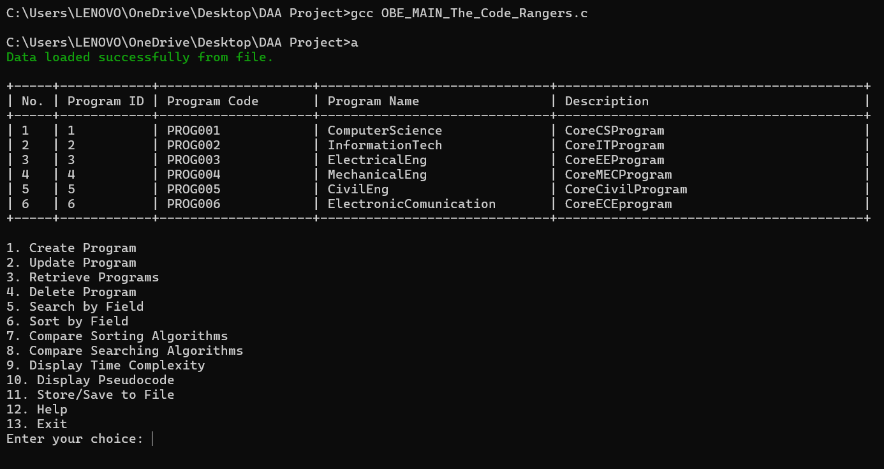
**Disadvantages:** Not applicable to unsorted datasets unless sorting is applied first.

**Time Complexity**:

**Linear Search**: O(n)

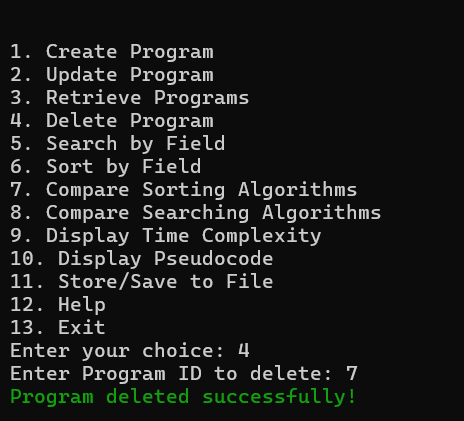
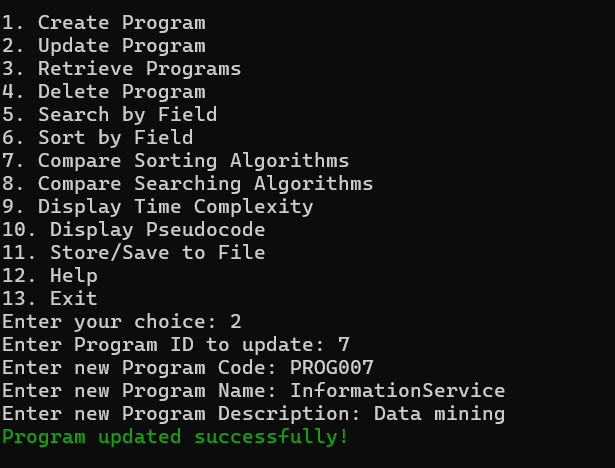
**Binary Search:** O(logn)(when data is sorted)

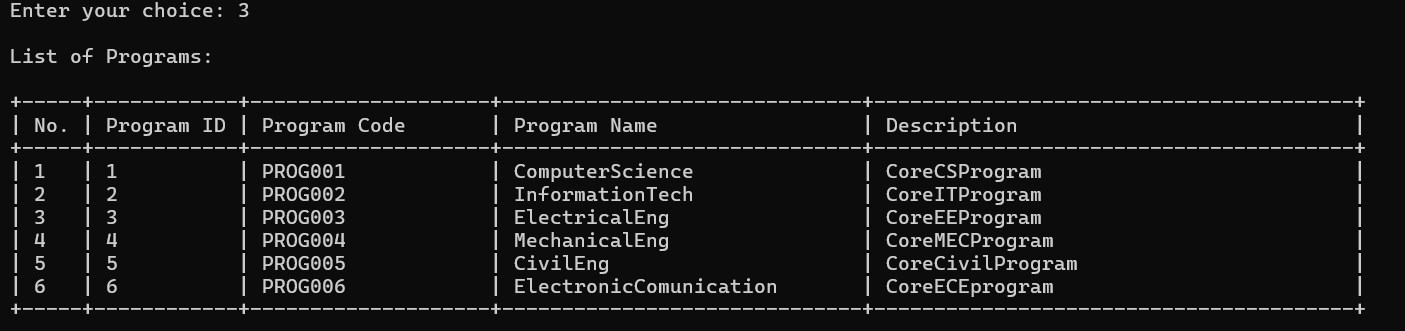
# **Screen Shots**

***OUTPUT***  ***CREATE***

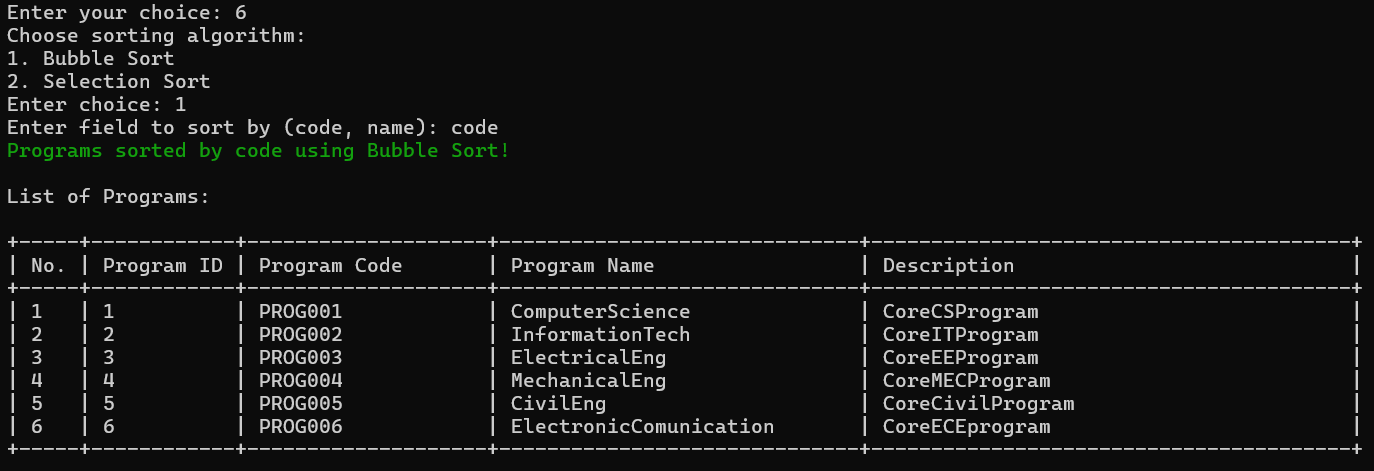
# 

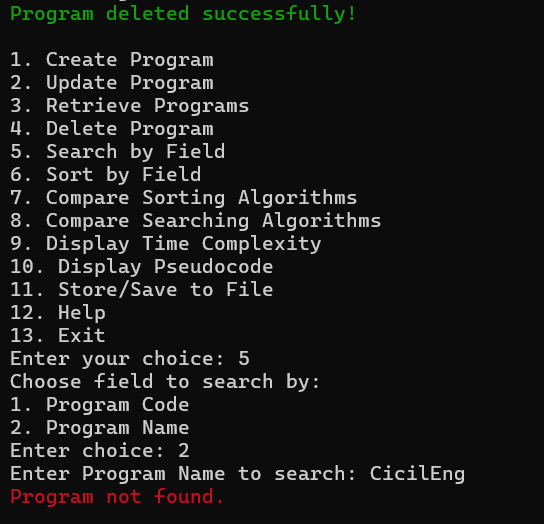
***UPDATE***  ***DELETE***



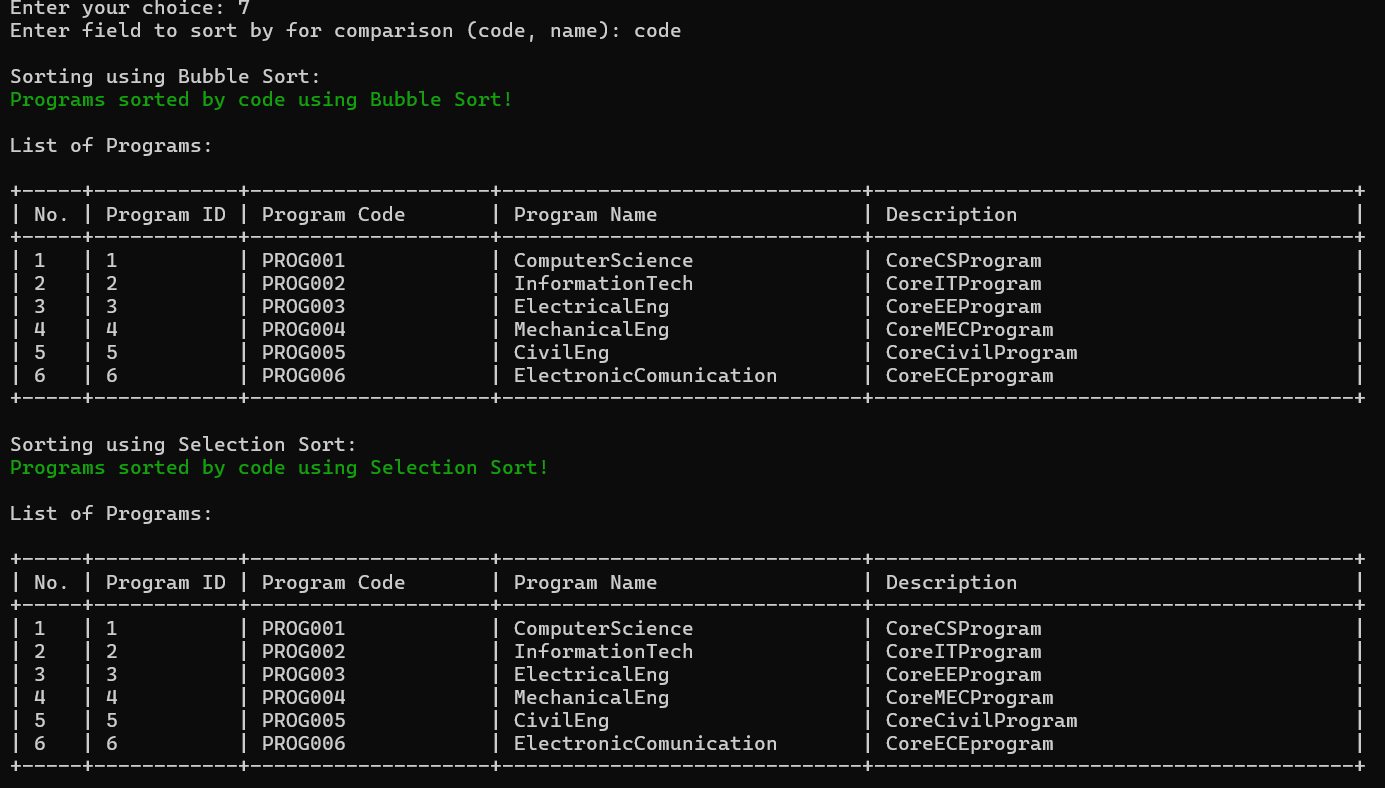


***RETRIEVE***

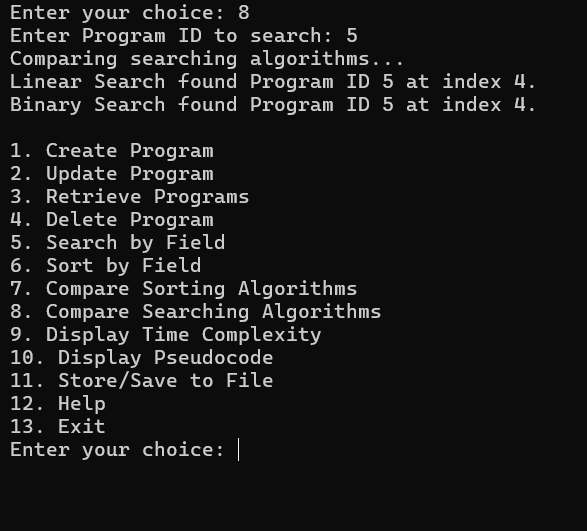
***SORTING***



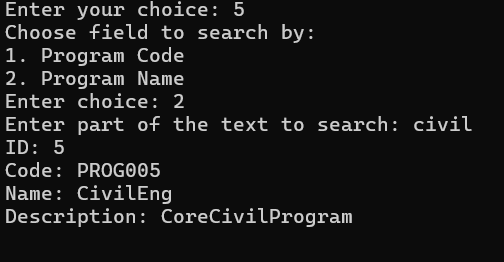
***SEARCHING(other case)***

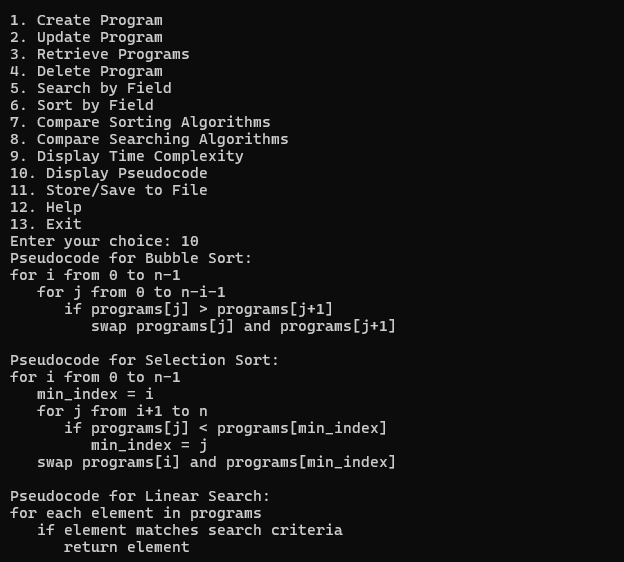
***COMPARISON ALGORITHM(Sorting)***

***COMPARISON ALGORITHM(Sorting)***

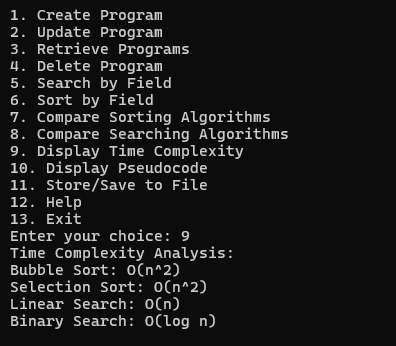


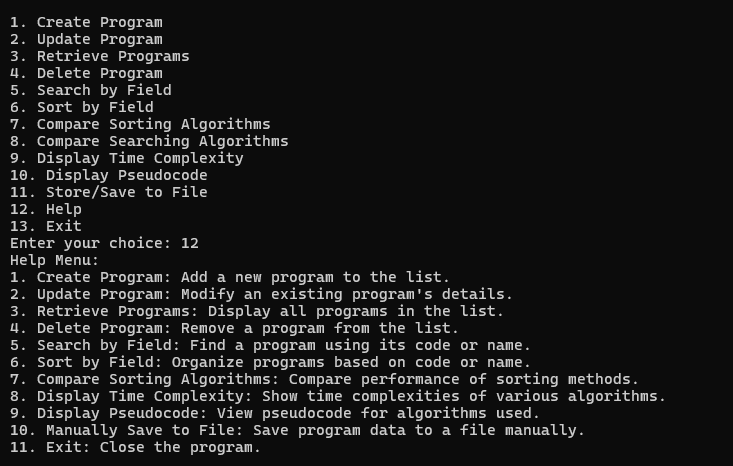
***SEARCHING USING SUBSTR***



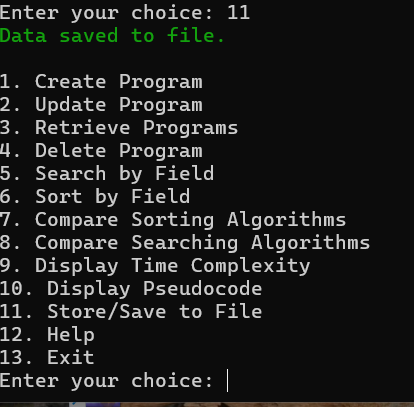
***PSEUDO CODE DISPLAY***

***DISPLAY TIME COMPLEXITY***



***HELP(for Documentation gudie)***

***STORING***



# **CONCLUSION**

The Program Management module provides an effective solution for managing program records, with functionalities for CRUD operations and efficient data retrieval via sorting and searching. By comparing sorting and searching algorithms, this project demonstrates the importance of choosing the right algorithm for performance optimization. The module’s modular structure, alongside file-based data persistence, ensures a reliable and maintainable system for program management.

THANK YOU!