# **TITLE:** DEPARTMENT

**TEAM NAME**: DEFENDERS

**TEAM MEMBERS**:

**D. Nikhil[AP23110011093]**

**A. Kiran[AP23110011126]**

**K. Praneeth[AP23110011123]**

**T. Manikanta[AP23110011132]**

**G. Venkatesh[AP23110011099]**

*A report for the CS204:Design and Analysis of Algorithm project*

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**SRM UNIVERSITY AP::AMARAVATI**

# INDEX

**Introduction**

Project Modules:

**Architecture Diagram**

**Module Description**

Field/table details

Algorithm Details:

(i)Sorting (ii)Searching

(ii) Storing the details in a text file

**Source Code**

**Comparison of Sorting Algorithms**

**Comparison of Searching Algorithms**

**Screen Shots**

**Conclusion**

## Introduction:

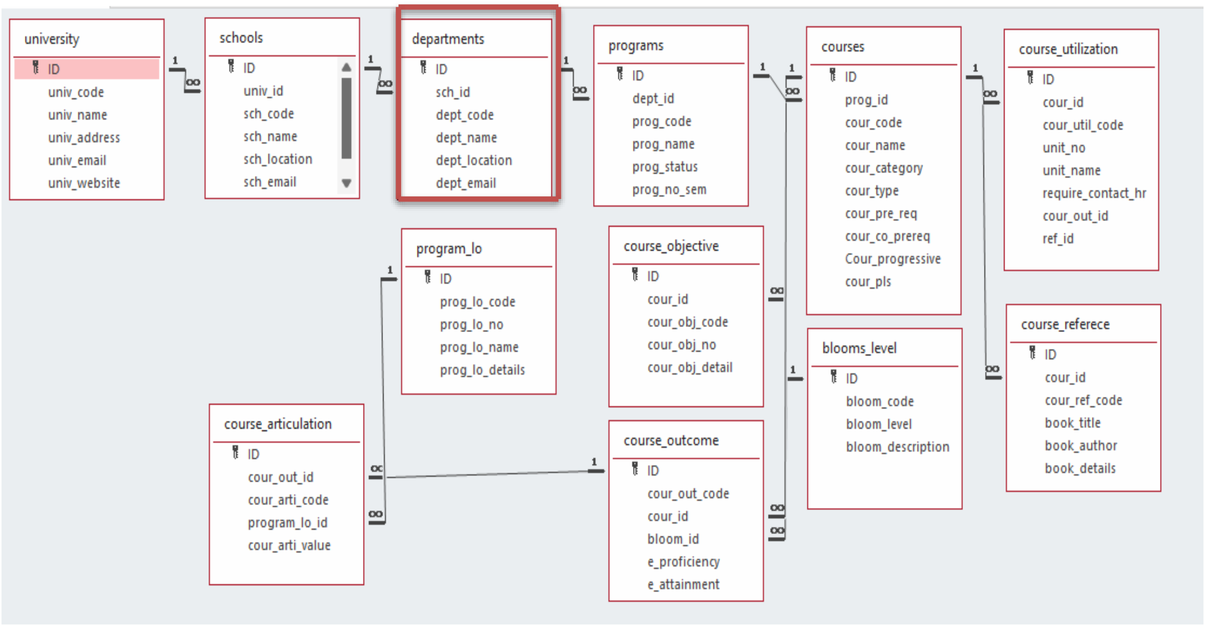
This project module manages records for various "departments," where each department is defined by its unique ID, name, code, and description. The module allows for essential CRUD (Create, Retrieve, Update, Delete) operations to efficiently manage department data. These functionalities include sorting and searching to facilitate quick access and organization. The information is stored in a text file for persistent data management, ensuring department records are maintained across sessions. This module highlights effective data handling practices, fundamental algorithms for sorting and searching, and the assessment of their efficiency to support structured data management within the project.

## Project Modules:

Various Modules available in the project are

1.Blooms Level setting
2.Program Level Objective Setting
3.University
4.Schools
5.Department
6.Programs
7.Courses
8.Course objective setting
9.Course Outcome Setting
10.Course Articulation matrix Setting
11.Course Utilization Setting
12.Course Reference Setting.

# Architecture Diagram



# Module Description

**Module Name:** .Department

**Module Description:**

This module is used to create,Update,Retrieve,Delete(hereafter known as CURD) details of the module and storing the details in the text file.you have to provide option for searching and sorting of fields mentioned below according to algorithms given for you

| **Field Name** | **Data type** |
|---|---|
| id | integer |
| sch_id | String |
| dept_code | String |
| dept_name | String |
| dept_location | String |
| dept_email | String |

# Algorithm Details:

## (i)Sorting

- You have to provide sorting based on **department_code ,department_name , department_email.**
- Compare the algorithm you have used with any of the other sorting algorithm
- Display the time complexity of both algorithms.
- Display the pseudocode/algorithm of the sorting algorithm used by you

## (ii)Searching

- You have provided sorting based on **department_code, department_name, department_email**
- Compare the algorithm used with any of the other algorithm you have learned
- Display the time complexity of both algorithms.
- Display the pseudocode/algorithm of the searching algorithm used by you.

## (iii)Storing the details in a text file

- Storing the details in the text file once details are entered.
- Delete the detail from the text file once details are deleted.
- Update the text file once details are updated.

# Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
// Department structure
typedef struct {
   int id;
   char sch_id[15];
   char dept_code[10];
   char dept_name[50];
   char dept_location[100];
   char dept_email[50];
} Department;
// Global array to store department data and a counter
Department departments[MAX];
int department_count = 0;
// File name for storing the details
const char* FILE_NAME = "department_setting.txt";
// Function declarations
void defenders_department_create();
void defenders_department_update();
void defenders_department_retrieve();
void defenders_department_delete();
void defenders_department_storing();
void defenders_department_sortbycode();
void defenders_department_searchbycode();
// Function to load data from the file into the departments array
void load_from_file() {
   FILE *file = fopen(FILE_NAME, "r");
   if (file == NULL) {
      return; // No file exists yet
   }
```

```
 department_count = 0;
    while (fscanf(file, "%d %s %s %s %s %s\n",
&departments[department_count].id,
    departments[department_count].sch_id,
    departments[department_count].dept_code,
    departments[department_count].dept_name,
    departments[department_count].dept_location,
    departments[department_count].dept_email) == 6) {
        department_count++;
    }
    fclose(file);
}
// Function to save data to the file
void defenders_department_storing() {
    FILE *file = fopen(FILE_NAME, "w");
    if (file == NULL) {
        printf("Error opening file!\n");
        return;
    }
    for (int i = 0; i < department_count; i++) {
        fprintf(file, "%d %s %s %s %s %s\n", departments[i].id,
        departments[i].sch_id, departments[i].dept_code,
        departments[i].dept_name, departments[i].dept_location,
        departments[i].dept_email);
    }
    fclose(file);
}
// Function to create a university record
void defenders_department_create() {
    if (department_count >= MAX) {
        printf("Department list is full!\n");
        return;
    }
    Department d;
    printf("Enter Department ID: ");
    scanf("%d", &d.id);
    printf("Enter School ID: ");
    scanf("%s", d.sch_id);
    printf("Enter Department Code: ");
    scanf("%s", d.dept_code);
    printf("Enter Department Name: ");
    scanf("%s", d.dept_name);
    printf("Enter Department Location: ");
    scanf("%s", d.dept_location);
    printf("Enter Department Email: ");
    scanf("%s", d.dept_email);
    departments[department_count++] = d;
    defenders_department_storing();
    printf("Department created successfully!\n");
}
// Function to update a university record
void defenders_department_update() {
    int id;
    printf("Enter Department ID to update: ");
    scanf("%d", &id);
    for (int i = 0; i < department_count; i++) {
        if (departments[i].id == id) {
            printf("Enter new School ID: ");
            scanf("%s", departments[i].sch_id);
            printf("Enter new Department Code: ");
            scanf("%s", departments[i].dept_code);
            printf("Enter new Department Name: ");
            scanf("%s", departments[i].dept_name);
            printf("Enter new Department Address: ");
            scanf("%s", departments[i].dept_location);
            printf("Enter new Department Email: ");
            scanf("%s", departments[i].dept_email);
            defenders_department_storing();
            printf("Department updated successfully!\n");
            return;
        }
```

```
 }
    printf("Department with ID %d not found.\n", id);
}
// Function to retrieve all department records
void defenders_department_retrieve() {
    printf("\nList of Departments:\n");
    for (int i = 0; i < department_count; i++) {
        printf("ID: %d\nSchool_ID: %s\nCode: %s\nName: %s\nLocation:
%s\nEmail: %s\n\n",
        departments[i].id, departments[i].sch_id,
        departments[i].dept_code, departments[i].dept_name,
        departments[i].dept_location, departments[i].dept_email);
    }
}
// Function to delete a university record
void defenders_department_delete() {
    int id;
    printf("Enter Department ID to delete: ");
    scanf("%d", &id);
    for (int i = 0; i < department_count; i++) {
        if (departments[i].id == id) {
            for (int j = i; j < department_count - 1; j++) {
                departments[j] =departments[j + 1];
            }
            department_count--;
            defenders_department_storing();
            printf("Department deleted successfully!\n");
            return;
        }
    }
     printf("Department with ID %d not found.\n", id);
}
// Function to search department by code
void defenders_department_searchbycode() {
    char code[10];
    printf("Enter Department Code to search: ");
    scanf("%s", code);
 for (int i = 0; i < department_count; i++) {
        if (strcmp(departments[i].dept_code, code) == 0) {
            printf("ID: %d\nSchool_ID: %s\nCode: %s\nName: %s\nLocation:
%s\nEmail: %s\n\n",
            departments[i].id, departments[i].sch_id,
            departments[i].dept_code, departments[i].dept_name,
            departments[i].dept_location, departments[i].dept_email);
            return;
        }
    }
    printf("Department with code %s not found.\n", code);
}
// Bubble sort by department code
void defenders_department_sortbycode() {
    for (int i = 0; i < department_count - 1; i++) {
        for (int j = 0; j < department_count - i - 1; j++) {
            if (strcmp(departments[j].dept_code, departments[j +
1].dept_code) > 0) {
                Department temp = departments[j];
                departments[j] = departments[j + 1];
                departments[j + 1] = temp;
            }
        }
    }
    printf("Departments sorted by code!\n");
    defenders_department_retrieve();
}
int main() {
    load_from_file();
    int choice;
    while (1) {
        printf("\n1. Create Department\n2. Update Department\n3. Retrieve
Department\n4. Delete Department\n5. Search by Code\n6. Sort by
Code\n7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
 switch (choice) {
      case 1:
        defenders_department_create();
        break;
      case 2:
        defenders_department_update();
        break;
      case 3:
        defenders_department_retrieve();
        break;
      case 4:
        defenders_department_delete();
        break;
      case 5:
        defenders_department_searchbycode();
        break;
      case 6:
        defenders_department_sortbycode();
        break;
      case 7:
        exit(0);
      default:
        printf("Invalid choice!\n");
    }
  }
  return 0;
}
```

# Comparison of Sorting Algorithms:

## 1. Bubble Sort(Primary Algorithm):

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

**Disadvantage:** Poor performance for larger datasets due to O(n^2) time complexity.

## Algorithm for bubble sort():

```
function bubbleSort(arr)
    n = length(arr)
    for i from 0 to n-1
        for j from 0 to n-i-2
            if arr[j] > arr[j+1]
                swap arr[j] and arr[j+1]
    return arr
```

**Time complexity:**

**Bubble sort:** O(n^2)

## 2. Merge sort(Comparison Algorithm):

**Advantage:** Merge Sort is efficient and stable for larger datasets.
**Disadvantage:** It is slower for smaller datasets, and it uses more space.

## Algorithm for Merge Sort():

```
function mergeSort(arr)
    if length(arr) <= 1
        return arr

    mid = length(arr) / 2
    left = mergeSort(arr[0 to mid-1])
    right = mergeSort(arr[mid to end])

    return merge(left, right)

function merge(left, right)
    result = empty array
    while left is not empty and right is not empty
        if left[0] <= right[0]
            append left[0] to result
            remove left[0] from left
        else
            append right[0] to result
            remove right[0] from right

    while left is not empty
        append left[0] to result
        remove left[0] from left

    while right is not empty
        append right[0] to result
        remove right[0] from right

    return result
```

**Time Complexity:**

**Merge Sort:** O(nlogn)

# Comparison of Searching Algorithms:

## 1. Linear Search(Primary Algorithm):

**Advantage:** Works well for unsorted data, straightforward implement.

**Disadvantage:** Inefficient for large datasets as it checks each element.

## Algorithm for Linear Search():

```
function linearSearch(arr, target)
    for i from 0 to length(arr) - 1
        if arr[i] == target
            return i  // Return the index of the found element
    return -1  // Target not found
```

## Time Complexity:

**Linear Search:** O(n)

## 2. Binary Search(Comparison Algorithm):

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

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

## Algorithm for Binary Search():

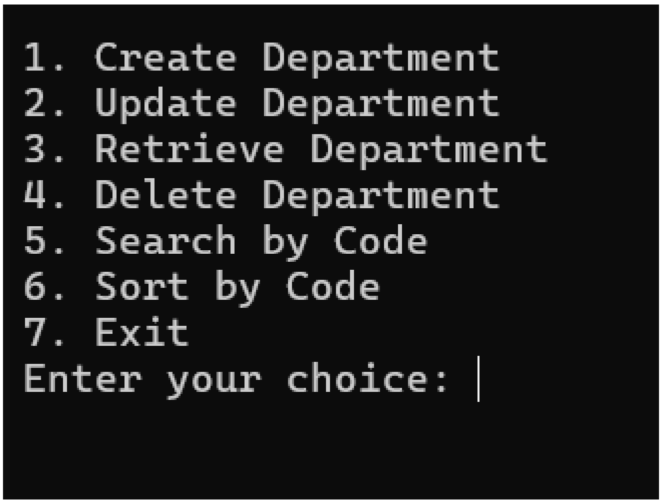
```
function binarySearch(arr, target)
    left = 0
    right = length(arr) - 1

    while left <= right
        mid = left + (right - left) / 2  // Prevents overflow for large arrays

        if arr[mid] == target
            return mid  // Target found
        else if arr[mid] < target
            left = mid + 1  // Search in the right half
        else
            right = mid - 1  // Search in the left half

    return -1  // Target not found
```

## Time complexity:

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

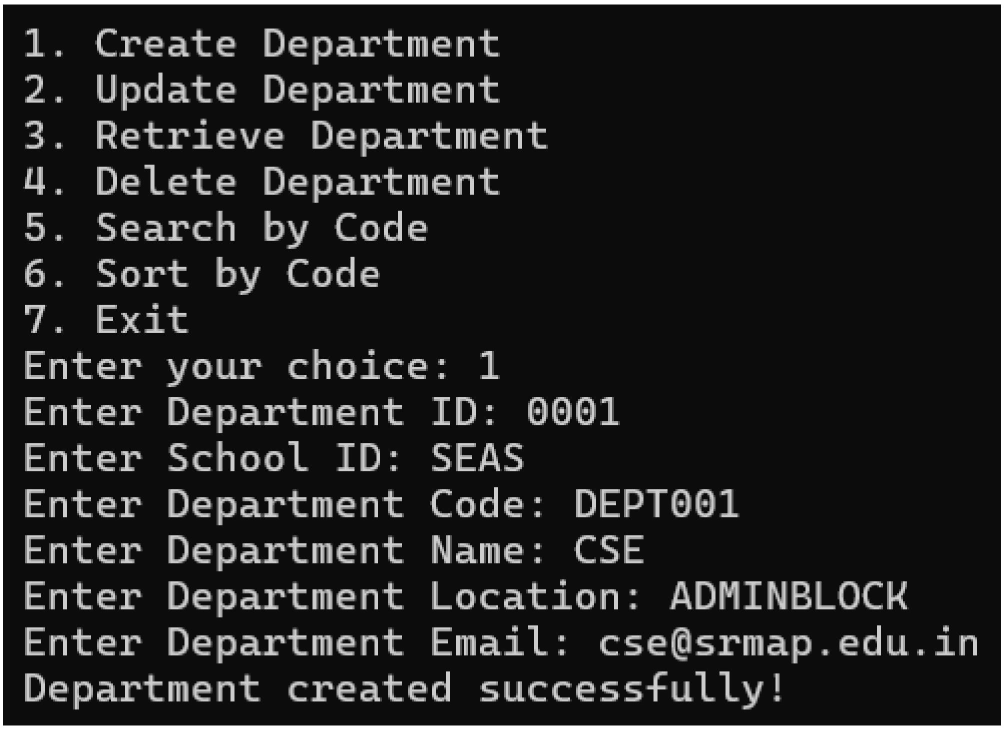
Screen Shots

## Output:



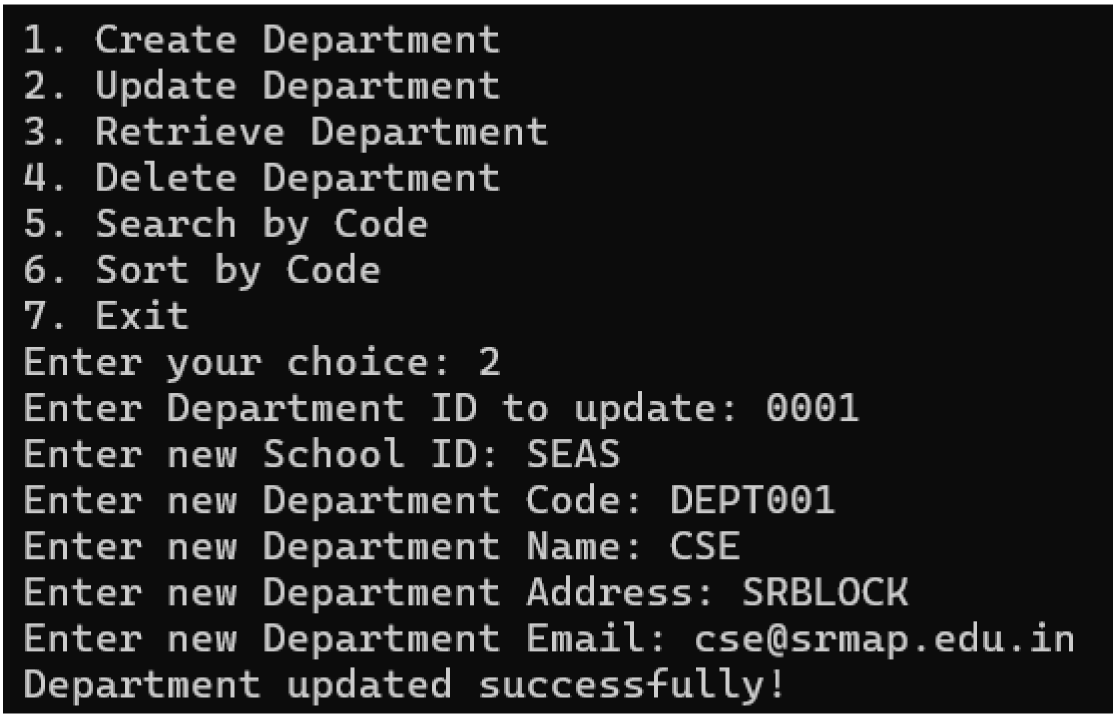
```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 
```

## Create:



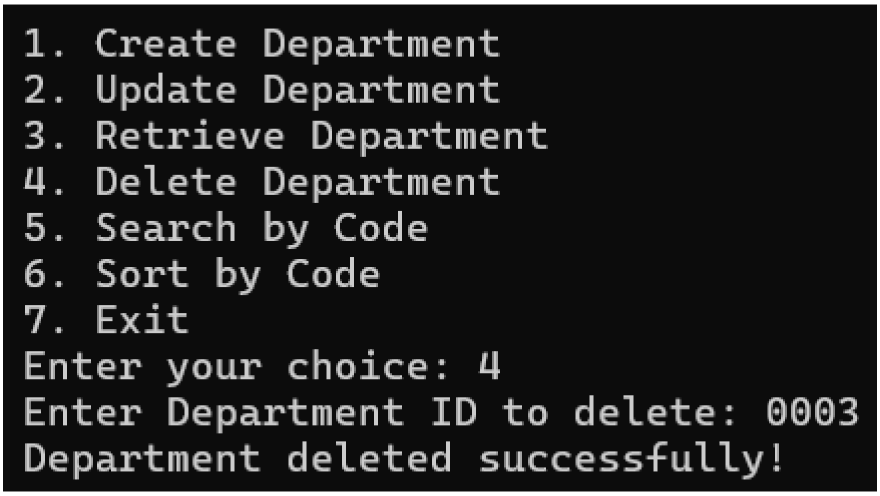
```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 1
Enter Department ID: 0001
Enter School ID: SEAS
Enter Department Code: DEPT001
Enter Department Name: CSE
Enter Department Location: ADMINBLOCK
Enter Department Email: cse@srmap.edu.in
Department created successfully!
```

## Update:



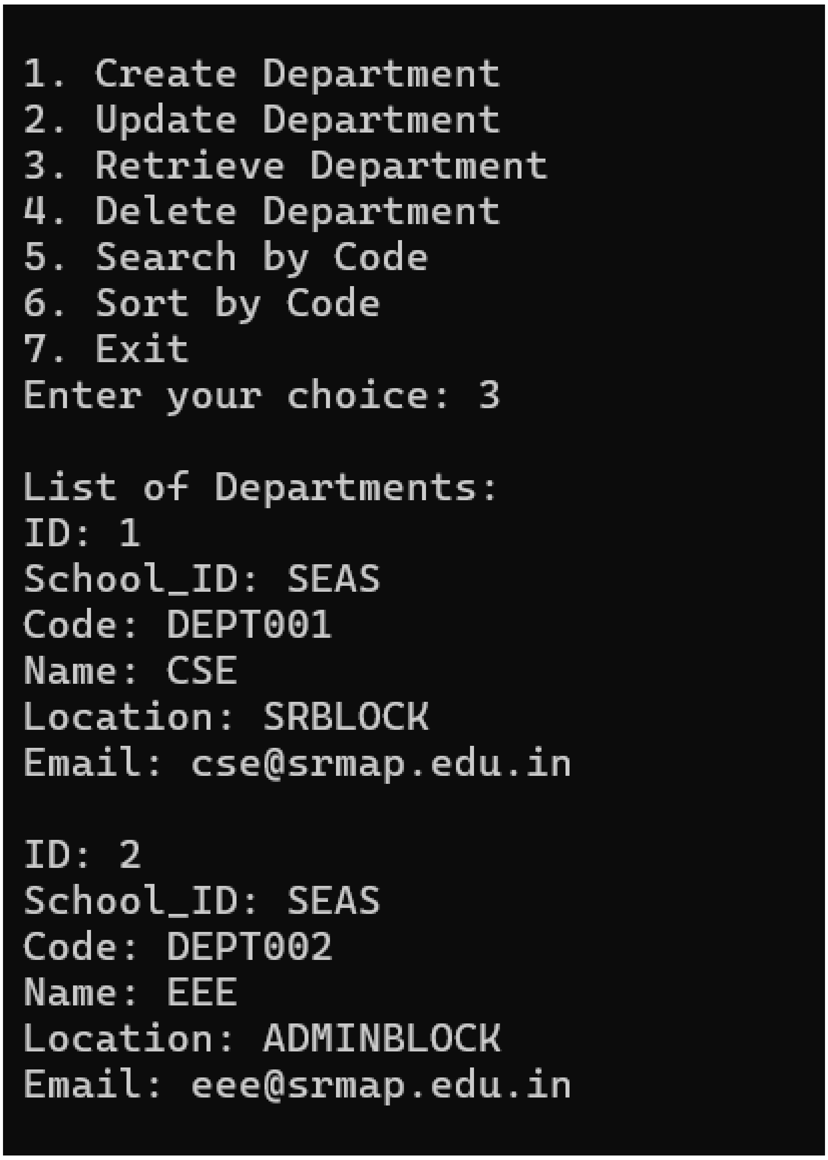
```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 2
Enter Department ID to update: 0001
Enter new School ID: SEAS
Enter new Department Code: DEPT001
Enter new Department Name: CSE
Enter new Department Address: SRBLOCK
Enter new Department Email: cse@srmap.edu.in
Department updated successfully!
```

## Delete:



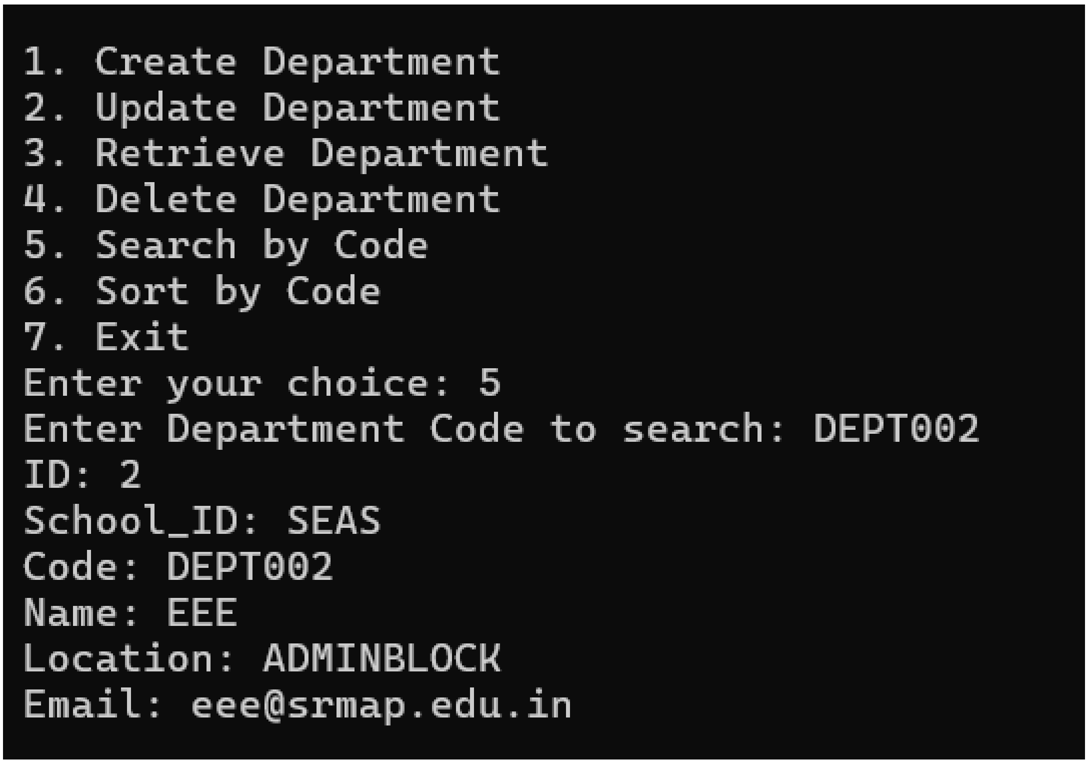
```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 4
Enter Department ID to delete: 0003
Department deleted successfully!
```

# Retrieve:



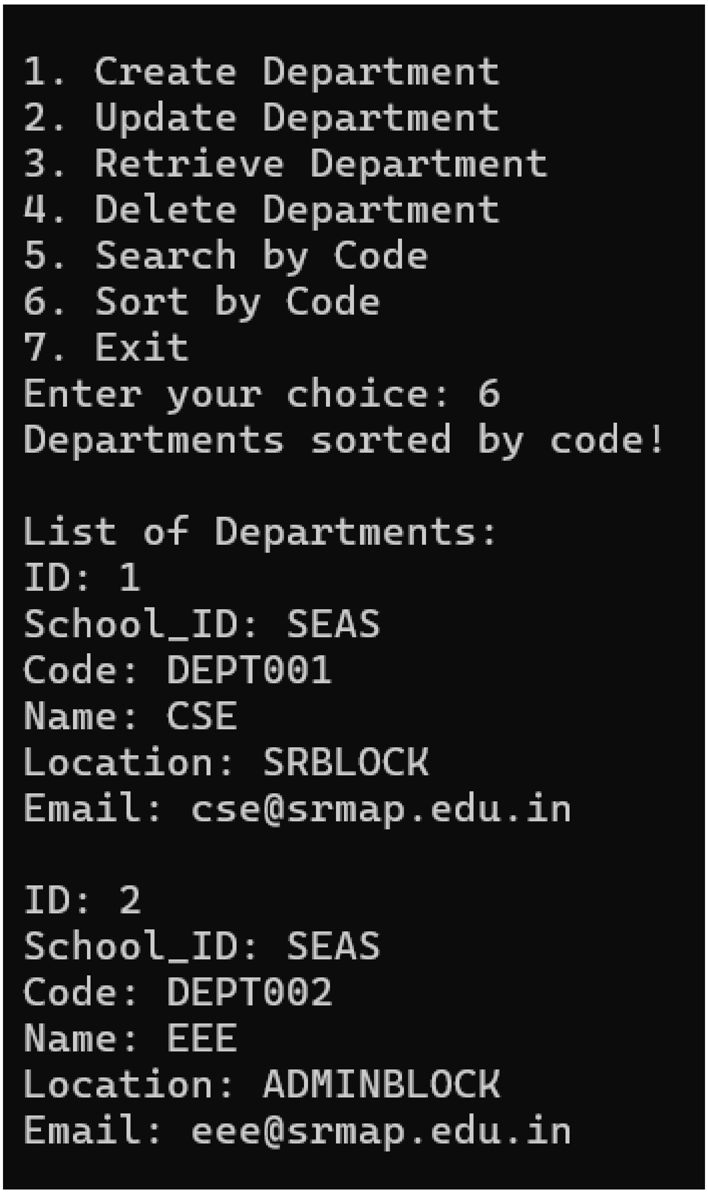
```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 3

List of Departments:
ID: 1
School_ID: SEAS
Code: DEPT001
Name: CSE
Location: SRBLOCK
Email: cse@srmap.edu.in

ID: 2
School_ID: SEAS
Code: DEPT002
Name: EEE
Location: ADMINBLOCK
Email: eee@srmap.edu.in
```

# Search by code:



```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 5
Enter Department Code to search: DEPT002
ID: 2
School_ID: SEAS
Code: DEPT002
Name: EEE
Location: ADMINBLOCK
Email: eee@srmap.edu.in
```

# Sort by code:



```
1. Create Department
2. Update Department
3. Retrieve Department
4. Delete Department
5. Search by Code
6. Sort by Code
7. Exit
Enter your choice: 6
Departments sorted by code!

List of Departments:
ID: 1
School_ID: SEAS
Code: DEPT001
Name: CSE
Location: SRBLOCK
Email: cse@srmap.edu.in

ID: 2
School_ID: SEAS
Code: DEPT002
Name: EEE
Location: ADMINBLOCK
Email: eee@srmap.edu.in
```

# Conclusion:

Upon performing CRUD (Create, Retrieve, Update, Delete) operations, the "Departments" module demonstrates its effectiveness as a reliable system for managing departmental data within the project. The implementation of these operations ensures that users can effortlessly add new departments, access and review existing data, modify records as needed, and remove outdated or incorrect entries. The sorting and searching features enhance data accessibility and organization, facilitating quick retrieval and decision-making.

By storing data in a text file for persistent access, the module provides a robust method to maintain data consistency across sessions. The successful execution of these operations highlights the module's capacity to streamline data management, improve data integrity, and support the overall efficiency of the project. This module stands as a practical application of essential data handling principles, demonstrating proficiency in implementing and managing structured data within an academic or organizational framework.