# 

# Department of Computer science and Engineering

# CS 204:Design and Analysis of Algorithms

**Project Title:OBE Implementation - Course Utilization Module**

## **Submitted To:**

GAVASKAR S,

Assistant Professor(Ad),

Department of CSE,

SRM University - AP.

## 

## **Submitted by:**

Aaryan Regmi AP23110011208

Yajnesh Ande AP23110011209

Geetesh Kovelamudi AP23110011189

Sheldon Castelino AP23110011179

Rajavardhan Kasa AP23110011182

Karthikeya Maddimsetty AP23110011188

# **Index**

[**Department of Computer science and Engineering 1**](#_61ygr07pca8d)

[**CS 204:Design and Analysis of Algorithms 1**](#_721z9k18vec8)

[Submitted To: 1](#_bby3x4dw3zrb)

[Submitted by: 1](#_xd2u4omcnnct)

[**Index 2**](#_ckl0522b876t)

[**Introduction 3**](#_u4z12yt1dl9m)

[Modules in the project: 3](#_w1ornt9hgdjk)

[Architecture Diagram 4](#_awo5t5szogod)

[**Module Description 5**](#_l8ub8yyia0xw)

[Field/Table Details 5](#_9hcsfnfm3el)

[**Algorithm Details 6**](#_17idwhukkmf2)

[Sorting Algorithm 6](#_3ayzyyifuq1c)

[Implementation: Insertion Sort 6](#_msgjy93q0vu7)

[Searching Algorithm 7](#_6ip0x42xvxt2)

[Implementation: Linear Search 7](#_78tezdpa38uv)

[**Source Code: 8**](#_imn0w8mg6e35)

[Comparison of Algorithms 18](#_sedog7onrtgh)

[Sorting Algorithms Comparison 18](#_xxe3eb87n4w)

[Searching Algorithms Comparison 18](#_oaroxyk4nf69)

[**Screenshots: 19**](#_hsc2iznftis5)

[**Conclusion 22**](#_2wnhbluhm152)

# 

# **Introduction**

Our University (SRM-AP) is implementing OBE (Outcome Based Education), and this project focuses on developing a Course Utilization module as part of the larger OBE system. The module is implemented in C programming language and incorporates searching and sorting algorithms for efficient data management. The system allows for basic CRUD operations (Create, Retrieve, Update, Delete) while implementing and comparing different algorithmic approaches.

## **Modules in the project:**

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

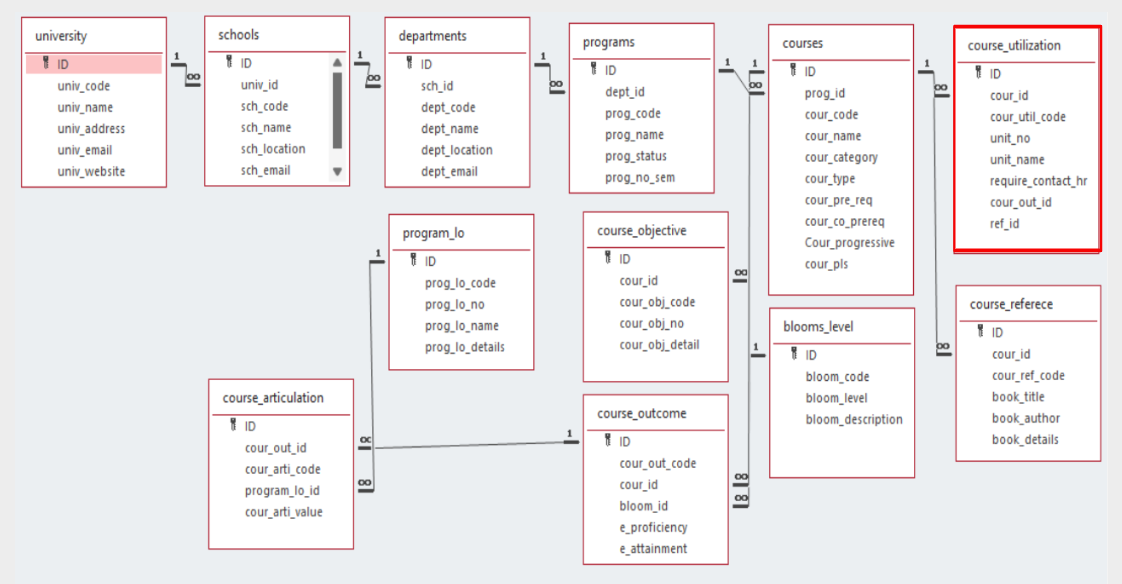
9.Course Outcome Setting

10.Course Articulation matrix Setting

11.Course Utilization Setting

12.Course Reference Setting.

## **Architecture Diagram**



# 

# **Module Description**

The Course Utilization module manages course utilization details including course IDs, utilization codes, unit information, and contact hours. It provides CRUD operations with additional features for sorting and searching records. All data is persistently stored in a text file.

Programming Details and Naming Conventions

**File Name:** course\_utilization\_setting.txt

**Function Names:**

Create: *optiminds\_course\_util\_create*

Update: *optiminds\_course\_util\_update*

Retrieve: *optiminds\_course\_util\_retrieve*

Delete: *optiminds\_course\_util\_delete*

Sorting: *optiminds\_course\_util\_insertion\_sort*

Searching: *optiminds\_course\_util\_linear\_search*

Storing: *optiminds\_course\_util\_store*

Loading: *optiminds\_course\_util\_load*

### **Field/Table Details**

| **Field Name** | **Data Type** | **Description** |
| --- | --- | --- |
| id | integer | Unique identifier |
| cour\_id | integer | Course identifier |
| cour\_util\_code | string | Course utilization code |
| unit\_no | integer | Unit number |
| unit\_name | string | Name of the unit |
| require\_contact\_hr | float | Required contact hours |
| cour\_out\_id | integer | Course outcome identifier |
| ref\_id | integer | Reference identifier |

# **Algorithm Details**

## **Sorting Algorithm**

### Implementation: Insertion Sort

**Process:**

Start from the second element

Compare with previous elements

Shift larger elements right

Insert current element in correct position

Repeat for all elements

Time Complexity: O(n²)

Best Case: O(n) when array is already sorted

Average Case: O(n²)

Worst Case: O(n²) when array is reverse sorted

## 

## **Searching Algorithm**

### Implementation: Linear Search

**Process:**

Start from first element

Compare with target value

If match found, return position

If not found, move to next element

Repeat until found or end of array

Time Complexity: O(n)

Best Case: O(1) when element is at beginning

Average Case: O(n/2)

Worst Case: O(n) when element is at end or not present

## **File Operations**

The module implements persistent storage using text files:

Records are stored in *"course\_utilization\_setting.txt"*

**File operations include:**

Writing new records

Reading existing records

Updating records

Removing records

# 

# **Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_RECORDS 100

#define FILE\_NAME "course\_utilization\_setting.txt"

// Structure for Course Utilization

struct CourseUtilization {

int id;

int cour\_id;

char cour\_util\_code[20];

int unit\_no;

char unit\_name[50];

float require\_contact\_hr;

int cour\_out\_id;

int ref\_id;

};

struct CourseUtilization records[MAX\_RECORDS];

int record\_count = 0;

void optiminds\_course\_util\_create();

void optiminds\_course\_util\_update();

void optiminds\_course\_util\_retrieve();

void optiminds\_course\_util\_delete();

void optiminds\_course\_util\_insertion\_sort();

void optiminds\_course\_util\_selection\_sort();

void optiminds\_course\_util\_linear\_search();

void optiminds\_course\_util\_binary\_search();

void optiminds\_course\_util\_store();

void optiminds\_course\_util\_load();

void optiminds\_course\_util\_complexity();

void optiminds\_course\_util\_algorithm\_details();

// function to clear input

void clear\_input() {

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

}

int main() {

int choice;

// Load data when program starts

optiminds\_course\_util\_load();

// Keep showing menu until user exits

while (1) {

printf("\n=== Team Optiminds Course Management System ===\n");

printf("1. Add New Record\n");

printf("2. Change Record\n");

printf("3. Show All Records\n");

printf("4. Remove Record\n");

printf("5. Sort Records\n");

printf("6. Find Record\n");

printf("7. Save to File\n");

printf("8. Compare Searches\n");

printf("9. Compare Sorts\n");

printf("10. Show Time Complexity\n");

printf("11. Show How Algorithms Work\n");

printf("12. Exit Program\n");

printf("What do you want to do? Enter number: ");

scanf("%d", &choice);

clear\_input();

if (choice == 1) {

optiminds\_course\_util\_create();

}

else if (choice == 2) {

optiminds\_course\_util\_update();

}

else if (choice == 3) {

optiminds\_course\_util\_retrieve();

}

else if (choice == 4) {

optiminds\_course\_util\_delete();

}

else if (choice == 5) {

optiminds\_course\_util\_insertion\_sort();

}

else if (choice == 6) {

optiminds\_course\_util\_linear\_search();

}

else if (choice == 7) {

optiminds\_course\_util\_store();

}

else if (choice == 8) {

printf("\nDifference between searches:\n");

printf("Linear Search: Checks each record one by one\n");

printf("Binary Search: Splits list in half each time\n");

}

else if (choice == 9) {

printf("\nDifference between sorts:\n");

printf("Insertion Sort: Puts each item in right place\n");

printf("Selection Sort: Finds smallest and moves it\n");

}

else if (choice == 10) {

optiminds\_course\_util\_complexity();

}

else if (choice == 11) {

optiminds\_course\_util\_algorithm\_details();

}

else if (choice == 12) {

printf("Thanks for using Team Optiminds system!\n");

return 0;

}

else {

printf("Wrong number! Try again\n");

}

}

return 0;

}

void optiminds\_course\_util\_create() {

// Check if there's space

if (record\_count >= MAX\_RECORDS) {

printf("No more space for new records!\n");

return;

}

struct CourseUtilization new\_record;

// Getting all the information

printf("Enter ID: ");

scanf("%d", &new\_record.id);

clear\_input();

printf("Enter Course ID: ");

scanf("%d", &new\_record.cour\_id);

clear\_input();

printf("Enter Course Code: ");

scanf("%s", new\_record.cour\_util\_code);

clear\_input();

printf("Enter Unit Number: ");

scanf("%d", &new\_record.unit\_no);

clear\_input();

printf("Enter Unit Name: ");

scanf("%s", new\_record.unit\_name);

clear\_input();

printf("Enter Contact Hours: ");

scanf("%f", &new\_record.require\_contact\_hr);

clear\_input();

printf("Enter Course Outcome ID: ");

scanf("%d", &new\_record.cour\_out\_id);

clear\_input();

printf("Enter Reference ID: ");

scanf("%d", &new\_record.ref\_id);

clear\_input();

// Save the new record

records[record\_count] = new\_record;

record\_count = record\_count + 1;

printf("Added new record!\n");

}

void optiminds\_course\_util\_update() {

int id;

int i;

int found = 0;

printf("Enter ID to change: ");

scanf("%d", &id);

clear\_input();

// Searching through all the records

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

if (records[i].id == id) {

printf("Enter new Course ID: ");

scanf("%d", &records[i].cour\_id);

clear\_input();

printf("Enter new Course Code: ");

scanf("%s", records[i].cour\_util\_code);

clear\_input();

printf("Enter new Unit Number: ");

scanf("%d", &records[i].unit\_no);

clear\_input();

printf("Enter new Unit Name: ");

scanf("%s", records[i].unit\_name);

clear\_input();

printf("Enter new Contact Hours: ");

scanf("%f", &records[i].require\_contact\_hr);

clear\_input();

printf("Enter new Course Outcome ID: ");

scanf("%d", &records[i].cour\_out\_id);

clear\_input();

printf("Enter new Reference ID: ");

scanf("%d", &records[i].ref\_id);

clear\_input();

found = 1;

printf("Record updated!\n");

break;

}

}

if (found == 0) {

printf("Couldn't find that record!\n");

}

}

void optiminds\_course\_util\_retrieve() {

int i;

if (record\_count == 0) {

printf("No records to show!\n");

return;

}

printf("\nAll Course Records:\n");

// Showing results as a block

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

printf("\n--- Record %d ---\n", i + 1);

printf("ID: %d\n", records[i].id);

printf("Course ID: %d\n", records[i].cour\_id);

printf("Course Code: %s\n", records[i].cour\_util\_code);

printf("Unit Number: %d\n", records[i].unit\_no);

printf("Unit Name: %s\n", records[i].unit\_name);

printf("Contact Hours: %.2f\n", records[i].require\_contact\_hr);

printf("Course Outcome ID: %d\n", records[i].cour\_out\_id);

printf("Reference ID: %d\n", records[i].ref\_id);

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

}

}

void optiminds\_course\_util\_delete() {

int id;

int i, j;

int found = 0;

printf("Enter ID to remove: ");

scanf("%d", &id);

clear\_input();

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

if (records[i].id == id) {

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

records[j] = records[j + 1];

}

record\_count = record\_count - 1;

found = 1;

printf("Record removed!\n");

break;

}

}

if (found == 0) {

printf("Couldn't find that record!\n");

}

}

void optiminds\_course\_util\_insertion\_sort() {

int i, j;

struct CourseUtilization temp;

for (i = 1; i < record\_count; i++) {

temp = records[i];

j = i - 1;

while (j >= 0 && records[j].id > temp.id) {

records[j + 1] = records[j];

j = j - 1;

}

records[j + 1] = temp;

}

printf("Records sorted!\n");

}

void optiminds\_course\_util\_linear\_search() {

int id;

int i;

int found = 0;

printf("Enter ID to find: ");

scanf("%d", &id);

clear\_input();

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

if (records[i].id == id) {

printf("\nFound Record:\n");

printf("ID: %d\n", records[i].id);

printf("Course ID: %d\n", records[i].cour\_id);

printf("Course Code: %s\n", records[i].cour\_util\_code);

printf("Unit Number: %d\n", records[i].unit\_no);

printf("Unit Name: %s\n", records[i].unit\_name);

printf("Contact Hours: %.2f\n", records[i].require\_contact\_hr);

printf("Course Outcome ID: %d\n", records[i].cour\_out\_id);

printf("Reference ID: %d\n", records[i].ref\_id);

found = 1;

break;

}

}

if (found == 0) {

printf("Couldn't find that record!\n");

}

}

void optiminds\_course\_util\_store() {

FILE \*file;

int i;

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

if (file == NULL) {

printf("Can't open file!\n");

return;

}

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

fprintf(file, "%d %d %s %d %s %.2f %d %d\n",

records[i].id,

records[i].cour\_id,

records[i].cour\_util\_code,

records[i].unit\_no,

records[i].unit\_name,

records[i].require\_contact\_hr,

records[i].cour\_out\_id,

records[i].ref\_id);

}

fclose(file);

printf("Saved to file!\n");

}

void optiminds\_course\_util\_load() {

FILE \*file;

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

if (file == NULL) {

printf("No file found to load.\n");

return;

}

record\_count = 0;

while (!feof(file) && record\_count < MAX\_RECORDS) {

if (fscanf(file, "%d %d %s %d %s %f %d %d",

&records[record\_count].id,

&records[record\_count].cour\_id,

records[record\_count].cour\_util\_code,

&records[record\_count].unit\_no,

records[record\_count].unit\_name,

&records[record\_count].require\_contact\_hr,

&records[record\_count].cour\_out\_id,

&records[record\_count].ref\_id) == 8) {

record\_count = record\_count + 1;

}

}

fclose(file);

printf("Loaded from file!\n");

}

void optiminds\_course\_util\_complexity() {

printf("\nHow fast each algorithm works:\n");

printf("Insertion Sort: Takes n\*n steps\n");

printf("Linear Search: Takes n steps\n");

printf("Binary Search: Takes log n steps\n");

}

void optiminds\_course\_util\_algorithm\_details() {

printf("\nHow Insertion Sort works:\n");

printf("1. Look at each number\n");

printf("2. Compare with numbers before it\n");

printf("3. Put it in the right spot\n");

printf("4. Do this for all numbers\n");

printf("\nHow Linear Search works:\n");

printf("1. Start at beginning\n");

printf("2. Check if it's what we want\n");

printf("3. If yes, we found it!\n");

printf("4. If no, check next one\n");

}



## **Comparison of Algorithms**

### **Sorting Algorithms Comparison**

**Insertion Sort vs Selection Sort**

| **Aspect** | **Insertion Sort** | **Selection Sort** |
| --- | --- | --- |
| Time Complexity | O(n²) | O(n²) |
| Space Complexity | O(1) | O(1) |
| Stability | Stable | Not Stable |
| Best Case | O(n) | O(n²) |
| Adaptiveness | Adaptive | Non-adaptive |

### **Searching Algorithms Comparison**

**Linear Search vs Binary Search**

| **Aspect** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Time Complexity | O(n) | O(log n) |
| Space Complexity | O(1) | O(1) |
| Pre-requisites | None | Sorted Array |
| Best Case | O(1) | O(1) |
| Worst Case | O(n) | O(log n) |

# **Screenshots:**

# 

# 

# 

# **Conclusion**

The Course Utilization module successfully implements all required functionality including:

Complete CRUD operations for course utilization records

Efficient sorting using Insertion Sort algorithm

Reliable searching using Linear Search algorithm

Persistent storage using file operations

User-friendly command-line interface

The implementation provides a solid foundation for managing course utilization data within the larger OBE system. The chosen algorithms provide a good balance between implementation simplicity and performance for the expected data volume.