**Department of Computer science and Engineering**

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

**Project Title:OBE Implementation**

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

**Team Name :** Team Hackers

**Team project:** Course Articulation Matrix

**Team leader:** Nidhi Chandra-AP23110011200

**Team members:**

G.Akhil babu - AP23110011185

CH.Pradnish - AP23110011198

D.Abhiram - AP23110011166

P.Nidhi Chandra - AP23110011200

K.Akash - AP23110011162

***SUBMITTED TO***

**Designed by**

**GAVASKAR S,**

**Assistant Professor(Ad),**

**Department of CSE,**

**SRM University - AP.**

**INDEX**

**Introduction**

The Course Articulation Matrix is a critical framework within Outcome-Based Education (OBE) that aligns course objectives and outcomes with program-level goals. It ensures that each course contributes meaningfully to the overall learning outcomes expected from students in a program. By defining a matrix that maps specific course objectives to relevant program outcomes, institutions can measure how well individual courses are supporting broader educational goals.

**Modules in the project:** 1. Course Objective Definition

2. Program Outcome Identification

3. Course-Outcome Mapping

4. Matrix Validation and Review

5. Assessment Strategy Development

6. Continuous Improvement Process

7. Documentation and Reporting

**Module Description**

This module uses course articulation matrices to map and align course-level outcomes with program objectives, employing measurable outcomes and structured matrices to support curriculum coherence and continuous improvement.

**Sample coding Template**

Articulation\_matrix\_hackers

**ChatGPT Usage**

Program Generated by ChatGPT

Instruction to use ChatGPT or other LLM Models

**Introduction**

The Course Articulation Matrix is a critical framework within Outcome-Based Education (OBE) that aligns course objectives and outcomes with program-level goals. It ensures that each course contributes meaningfully to the overall learning outcomes expected from students in a program. By defining a matrix that maps specific course objectives to relevant program outcomes, institutions can measure how well individual courses are supporting broader educational goals.

**Project Module:**

1. Course Objective Definition

2. Program Outcome Identification

3. Course-Outcome Mapping

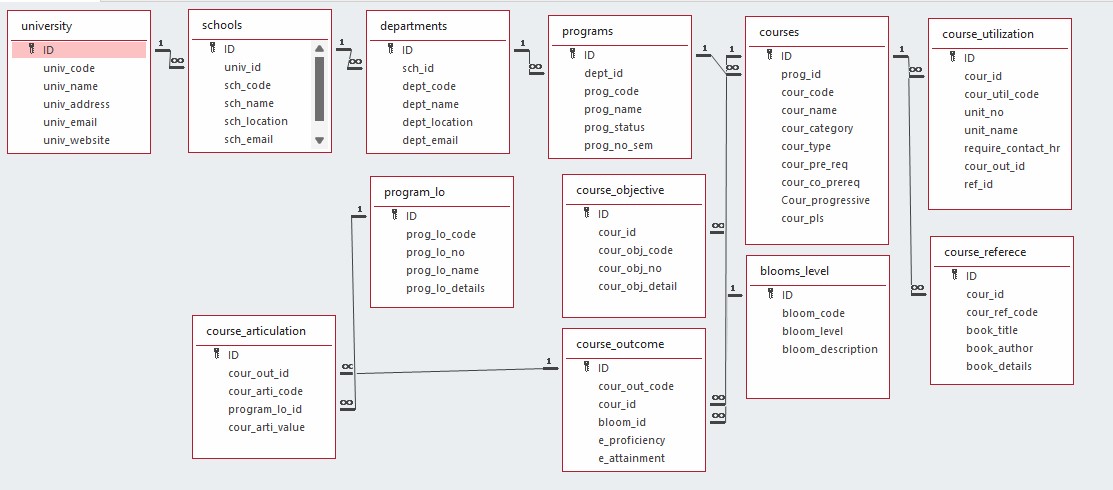
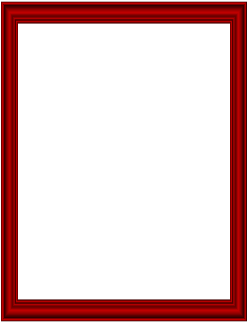
4. Matrix Validation and Review

5. Assessment Strategy Development

6. Continuous Improvement Process

7. Documentation and Reporting

**Architecture Diagram**



# 

# **Module Description**

Programming Details Naming Conventions to be Used:

• **File name**: TEAM\_HACKERS\_curriculum\_mapping

• Function/Method Name:

• **Create:**

TEAM\_HACKERS\_curriculum\_mapping\_create

• **Update**:

TEAM\_HACKERS\_curriculum\_mapping\_update

• **Retrieve**:

TEAM\_HACKERS\_curriculum\_mapping\_retrieve

• **Delete**:

TEAM\_HACKERS\_curriculum\_mapping\_delete

• **Sorting** (e.g., using Quick Sort to organize CLOs or PLOs):

TEAM\_HACKERS\_curriculum\_mapping\_quicksort

• **Searching** (e.g., searching for a specific CLO or PLO in the matrix using Binary Search):

TEAM\_HACKERS\_curriculum\_mapping\_binarysearch

• **Storing:**

TEAM\_HACKERS\_curriculum\_mapping\_storing

• Comparison (both Searching and Sorting):

• **For Searching:**

TEAM\_HACKERS\_curriculum\_mapping\_compare\_search\_binarysearch

• **For Sorting:**

TEAM\_HACKERS\_curriculum\_mapping\_compare\_sorting\_quicksort

• Time Complexity (both Searching and Sorting):

• **For Searching:**

TEAM\_HACKERS\_curriculum\_mapping\_complexity\_search

• **For Sorting**:

TEAM\_HACKERS\_curriculum\_mapping\_complexity\_sorting

• Algorithm Details (Pseudocode or Steps) (both Searching and Sorting):

• **For Searching:**

TEAM\_HACKERS\_curriculum\_mapping\_binarysearch\_details

• **For Sorting:**

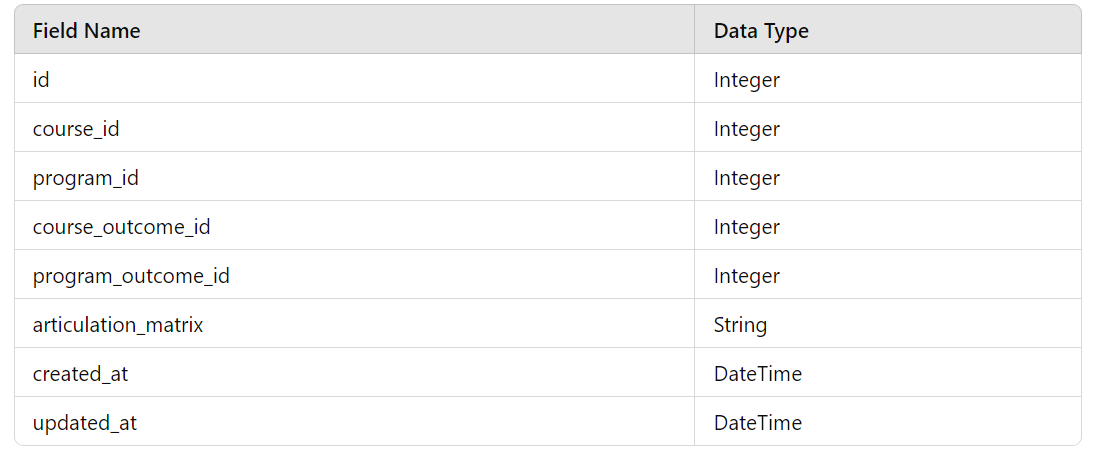
TEAM\_HACKERS\_curriculum\_mapping\_quicksort\_details

• File Name (for Storing the Details):

• File name to be used:

university\_setting.txt

Field/table details: For Programs



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:

**Source Code**

#include <stdio.h>

#define MAX\_COURSES 5

#define MAX\_OUTCOMES 5

// Function to input the matrix

void inputMatrix(int matrix[MAX\_COURSES][MAX\_OUTCOMES], int courses, int outcomes) {

printf("Enter the course articulation matrix:\n");

for (int i = 0; i < courses; i++) {

for (int j = 0; j < outcomes; j++) {

printf("Matrix[%d][%d]: ", i + 1, j + 1);

scanf("%d", &matrix[i][j]);

}

}

}

// Function to display the matrix

void displayMatrix(int matrix[MAX\_COURSES][MAX\_OUTCOMES], int courses, int outcomes) {

printf("\nCourse Articulation Matrix:\n");

for (int i = 0; i < courses; i++) {

for (int j = 0; j < outcomes; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

// Function to calculate the sum of a row

int rowSum(int matrix[MAX\_COURSES][MAX\_OUTCOMES], int row, int outcomes) {

int sum = 0;

for (int i = 0; i < outcomes; i++) {

sum += matrix[row][i];

}

return sum;

}

// Quick Sort Helper Functions

void swapRows(int matrix[MAX\_COURSES][MAX\_OUTCOMES], int row1, int row2, int outcomes) {

for (int i = 0; i < outcomes; i++) {

int temp = matrix[row1][i];

matrix[row1][i] = matrix[row2][i];

matrix[row2][i] = temp;

}

}

int partition(int matrix[MAX\_COURSES][MAX\_OUTCOMES], int low, int high, int outcomes) {

int pivot = rowSum(matrix, high, outcomes);

int i = low - 1;

for (int j = low; j < high; j++) {

if (rowSum(matrix, j, outcomes) <= pivot) {

i++;

swapRows(matrix, i, j, outcomes);

}

}

swapRows(matrix, i + 1, high, outcomes);

return i + 1;

}

int main() {

int courses, outcomes;

int target[MAX\_OUTCOMES];

printf("Enter the number of courses: ");

scanf("%d", &courses);

printf("Enter the number of outcomes: ");

scanf("%d", &outcomes);

int matrix[MAX\_COURSES][MAX\_OUTCOMES];

inputMatrix(matrix, courses, outcomes);

// Sorting using Quick Sort

quickSort(matrix, 0, courses - 1, outcomes);

printf("\nSorted ");

displayMatrix(matrix, courses, outcomes);

printf("Enter the course articulation to search for:\n");

for (int i = 0; i < outcomes; i++) {

printf("Target[%d]: ", i + 1);

scanf("%d", &target[i]);

}

// Searching using Linear Search

int linearIndex = linearSearch(matrix, courses, outcomes, target);

if (linearIndex != -1) {

printf("Course articulation found using Linear Search at index: %d\n", linearIndex + 1);

} else {

printf("Course articulation not found using Linear Search.\n");

}

// Searching using Binary Search

int binaryIndex = binarySearch(matrix, 0, courses - 1, outcomes, target);

if (binaryIndex != -1) {

printf("Course articulation found using Binary Search at index: %d\n", binaryIndex + 1);

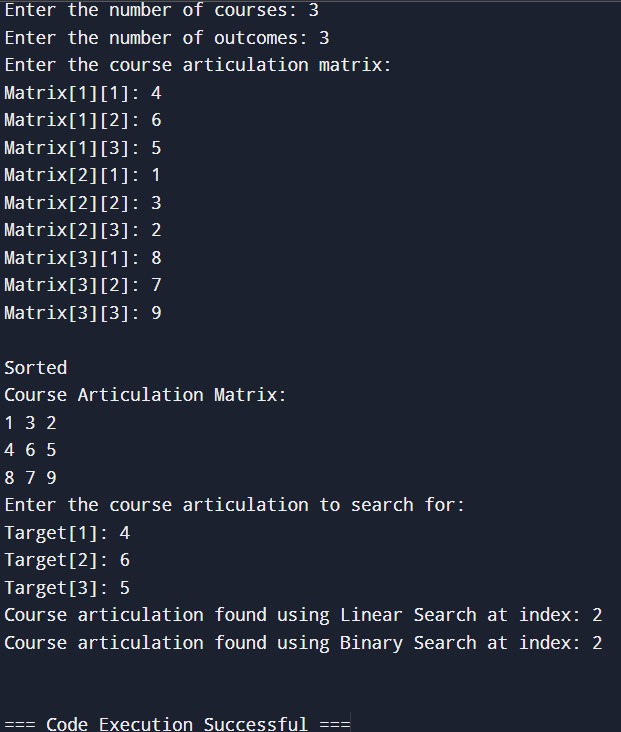
} else {

printf("Course articulation not found using Binary Search.\n");

}

return 0;

}

**Screenshots**

# **CONCLUSION**

In the programming section, we designed a comprehensive C program that handles a Course Articulation Matrix. This program includes functionalities for inputting the matrix, displaying it, sorting the courses based on specific criteria, and searching for a particular course articulation. We provided detailed steps to capture screenshots and explained how to document the code with an architecture diagram.

By implementing these concepts, you gained valuable insights into both discrete mathematics and practical programming, enhancing your problem-solving skills and ability to organize and present technical information effectively.

.

THANK YOU!