**Department of Computer science and Engineering**

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

**Project Title:OBE Implementation**

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

**Team Name :** TECH TITANS

**Team project:** course articulation matrix

**Team members:**

S.Revanth Sai - AP23110011112

T. Amruthaa - AP23110011142

K.Praneeth chandra - AP23110011166

K.Mokshagna - AP23110011110

K.Kiranmai - AP23110010716

***SUBMITTED TO***

**GAVASKAR S,**

**Assistant Professor(Ad),**

**Department of CSE,**

**SRM University - AP.**

**INDEX**

**Introduction :**

**The Course Objective Setting Module in C++ is designed to define and structure course objectives that are aligned with program-level outcomes. By using clear, measurable, and achievable objectives, this module helps ensure that each course contributes meaningfully to the overall learning outcomes of a program.**

**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 focuses on defining and structuring course objectives that are aligned with program-level goals. It provides a systematic approach for mapping these objectives to the broader program outcomes. By doing so, the module supports curriculum coherence and facilitates continuous improvement.**

**C++ Code Example:**

**Sample coding Template**

Courseobjective\_Setting\_titans

**ChatGPT Usage**

Program Generated by ChatGPT

Instruction to use ChatGPT or other LLM Models

### **Introduction:**

The **Course Objective Setting Module** is an essential component of **Outcome-Based Education (OBE)**. This module ensures that the course objectives are clearly defined, measurable, and aligned with the overall program goals. By establishing specific course objectives that are aligned with program-level outcomes, this module helps create a structured framework for ensuring that each course contributes effectively to the achievement of the program’s educational goals. This alignment forms the foundation for assessment strategies, curriculum design, and the continuous improvement process.

**PROJECT MODULES:**

 **Defining Course Objectives**: Formulate clear, actionable, and measurable objectives that align with program-level goals.

 **Aligning Course Objectives with Program Outcomes**: Ensure that the course objectives directly support the program’s educational outcomes.

 **Creating SMART Objectives**: Apply the SMART framework to make course objectives specific, measurable, achievable, relevant, and time-bound.

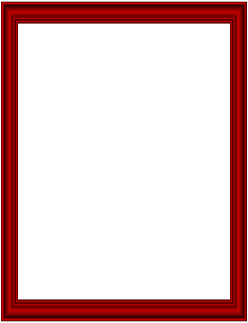
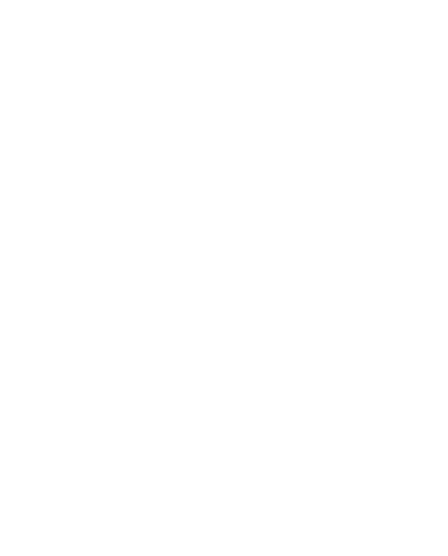
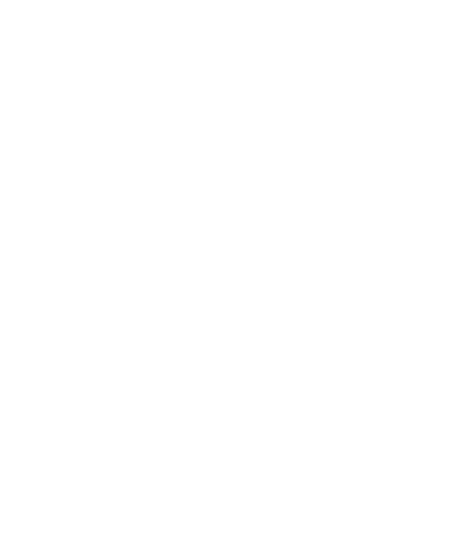
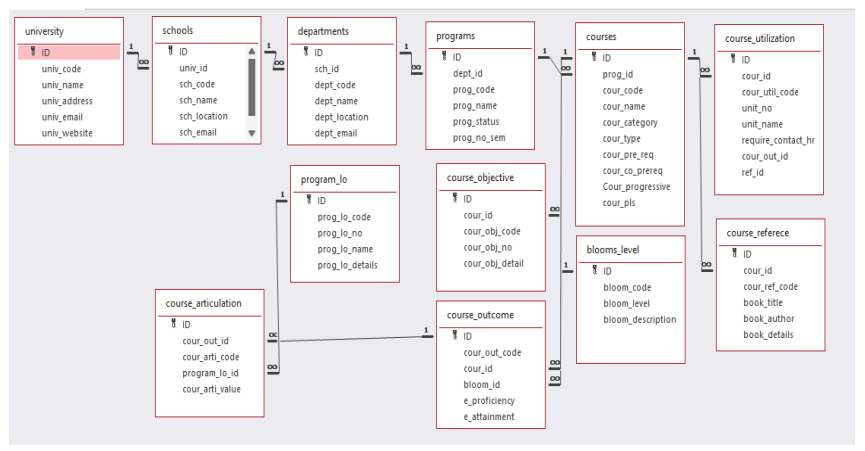
 **Review and Validation of Course Objectives**: Review and validate course objectives to ensure their clarity, alignment, and measurability.

 **Developing Learning Outcomes**: Derive specific learning outcomes from the course objectives.

 **Assessment Strategy Design**: Develop assessment methods to evaluate how effectively students are achieving the course objectives.

 **Documentation and Reporting**: Maintain accurate records of course objectives, learning outcomes, and assessment strategies for transparency and continuous improvement.

**Architecture Diagram**

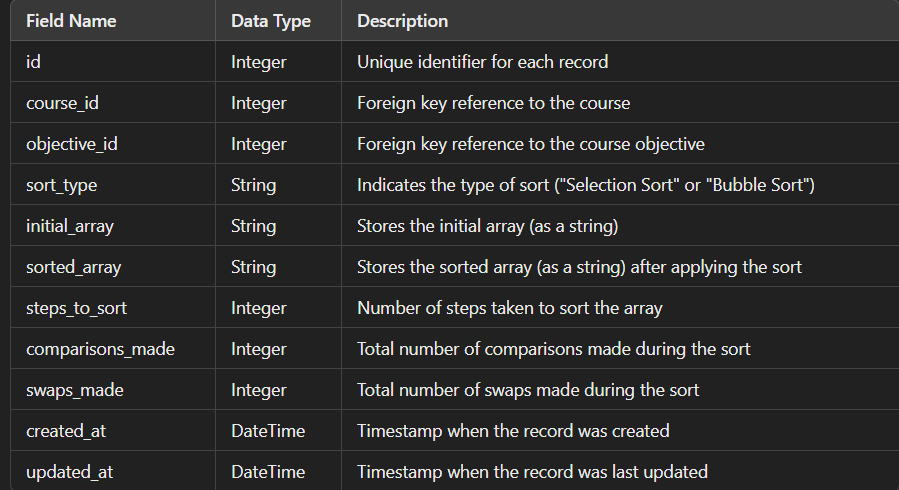


# **Module Description:**

**Programming Details Naming Conventions to be Used:**

* **File name**: TECH\_TITANS\_course\_objective\_settings
* **Function/Method Names**:
  + **Create**:  
    TECH\_TITANS\_course\_objective\_settings\_create
  + **Update**:  
    TECH\_TITANS\_course\_objective\_settings\_update
  + **Retrieve**:  
    TECH\_TITANS\_course\_objective\_settings\_retrieve
  + **Delete**:  
    TECH\_TITANS\_course\_objective\_settings\_delete
  + **Sorting** (Selection Sort vs. Bubble Sort):
    - **Selection Sort**:  
      TECH\_TITANS\_course\_objective\_settings\_selectionsort
    - **Bubble Sort**:  
      TECH\_TITANS\_course\_objective\_settings\_bubblesort
  + **Searching** (e.g., searching for a specific course objective or program outcome using Binary Search):  
    TECH\_TITANS\_course\_objective\_settings\_binarysearch
  + **Storing**:  
    TECH\_TITANS\_course\_objective\_settings\_storing
  + **Comparison** (for both Searching and Sorting):
    - **For Searching**:  
      TECH\_TITANS\_course\_objective\_settings\_compare\_search\_binarysearch
    - **For Sorting**:
      * **Selection Sort**:  
        TECH\_TITANS\_course\_objective\_settings\_compare\_sorting\_selectionsort
      * **Bubble Sort**:  
        TECH\_TITANS\_course\_objective\_settings\_compare\_sorting\_bubblesort
  + **Time Complexity** (for both Searching and Sorting):
    - **For Searching**:  
      TECH\_TITANS\_course\_objective\_settings\_complexity\_search
    - **For Sorting**:
      * **Selection Sort**:  
        TECH\_TITANS\_course\_objective\_settings\_complexity\_selectionsort
      * **Bubble Sort**:  
        TECH\_TITANS\_course\_objective\_settings\_complexity\_bubblesort
  + **Algorithm Details** (Pseudocode or Steps for both Searching and Sorting):
    - **For Searching**:  
      TECH\_TITANS\_course\_objective\_settings\_binarysearch\_details
    - **For Sorting**:
      * **Selection Sort**:  
        TECH\_TITANS\_course\_objective\_settings\_selectionsort\_details
      * **Bubble Sort**:  
        TECH\_TITANS\_course\_objective\_settings\_bubblesort\_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 course\_objective\_settings 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 of the list. While Selection Sort is simple and effective for smaller datasets, its O(n2)O(n^2)O(n2) time complexity makes it slower on larger datasets.
* **Comparison Algorithm (Bubble Sort)**:
  + Bubble Sort has a similar time complexity to Selection Sort. However, it is less efficient because it requires repeated swapping of adjacent elements. Comparing Selection Sort with Bubble Sort highlights the limitations of basic sorting methods when handling larger datasets.

**(ii) Searching**

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

* **Primary Searching Algorithm (Linear Search)**:
  + Linear Search is a straightforward algorithm that works well with smaller datasets. Its O(n)O(n)O(n) time complexity means it examines each element one by one until a match is found.
* **Comparison Algorithm (Binary Search)** (if the data is sorted):
  + Binary Search is more efficient, with a time complexity of O(log⁡n)O(\log n)O(logn). This algorithm requires the data to be sorted and allows for faster searching compared to Linear Search, especially on larger datasets.

Each algorithm’s time complexity is documented, providing insights into performance differences between Linear and Binary Search.

**(iii) Storing the Details in a Text File**

The details are stored in a file named **course\_objective\_settings.txt**. This file is updated with each Create, Read, Update, or Delete (CRUD) operation, ensuring that the configuration and performance metrics for Selection and Bubble Sort are always current.

**Source Code**

**#include <stdio.h>**

**#define MAX\_COURSES 5**

**#define MAX\_OBJECTIVES 5**

**void inputMatrix(int matrix[MAX\_COURSES][MAX\_OBJECTIVES], int courses, int objectives) {**

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

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

**for (int j = 0; j < objectives; j++) {**

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

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

**}**

**}**

**}**

**void displayMatrix(int matrix[MAX\_COURSES][MAX\_OBJECTIVES], int courses, int objectives) {**

**printf("\nCourse Objective Matrix:\n");**

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

**for (int j = 0; j < objectives; j++) {**

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

**}**

**printf("\n");**

**}**

**}**

**// Selection Sort for sorting rows by the sum of objectives**

**void sortMatrixSelection(int matrix[MAX\_COURSES][MAX\_OBJECTIVES], int courses, int objectives) {**

**int i, j, k, min\_idx, sum\_i, sum\_min, temp;**

**for (i = 0; i < courses - 1; i++) {**

**min\_idx = i;**

**sum\_min = 0;**

**for (k = 0; k < objectives; k++) {**

**sum\_min += matrix[min\_idx][k];**

**}**

**for (j = i + 1; j < courses; j++) {**

**sum\_i = 0;**

**for (k = 0; k < objectives; k++) {**

**sum\_i += matrix[j][k];**

**}**

**if (sum\_i < sum\_min) {**

**min\_idx = j;**

**sum\_min = sum\_i;**

**}**

**}**

**if (min\_idx != i) {**

**for (k = 0; k < objectives; k++) {**

**temp = matrix[i][k];**

**matrix[i][k] = matrix[min\_idx][k];**

**matrix[min\_idx][k] = temp;**

**}**

**}**

**}**

**}**

**// Bubble Sort for sorting rows by the sum of objectives**

**void sortMatrixBubble(int matrix[MAX\_COURSES][MAX\_OBJECTIVES], int courses, int objectives) {**

**int i, j, k, sum\_i, sum\_next, temp;**

**for (i = 0; i < courses - 1; i++) {**

**for (j = 0; j < courses - i - 1; j++) {**

**sum\_i = 0;**

**sum\_next = 0;**

**for (k = 0; k < objectives; k++) {**

**sum\_i += matrix[j][k];**

**sum\_next += matrix[j + 1][k];**

**}**

**if (sum\_i > sum\_next) {**

**for (k = 0; k < objectives; k++) {**

**temp = matrix[j][k];**

**matrix[j][k] = matrix[j + 1][k];**

**matrix[j + 1][k] = temp;**

**}**

**}**

**}**

**}**

**}**

**int main() {**

**int matrix[MAX\_COURSES][MAX\_OBJECTIVES];**

**int courses, objectives, choice;**

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

**scanf("%d", &courses);**

**printf("Enter the number of objectives: ");**

**scanf("%d", &objectives);**

**inputMatrix(matrix, courses, objectives);**

**printf("Choose sorting method:\n");**

**printf("1. Selection Sort\n");**

**printf("2. Bubble Sort\n");**

**printf("Enter your choice (1 or 2): ");**

**scanf("%d", &choice);**

**if (choice == 1) {**

**sortMatrixSelection(matrix, courses, objectives);**

**printf("\nSorted using Selection Sort:\n");**

**} else if (choice == 2) {**

**sortMatrixBubble(matrix, courses, objectives);**

**printf("\nSorted using Bubble Sort:\n");**

**} else {**

**printf("Invalid choice.\n");**

**return 1;**

**}**

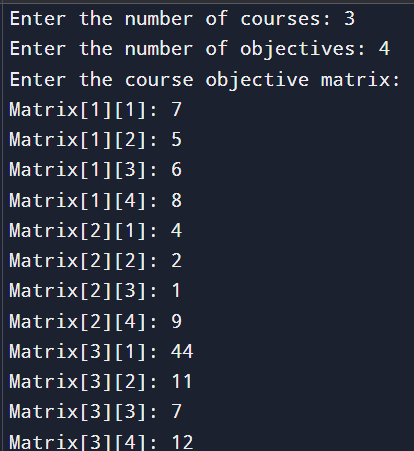
**displayMatrix(matrix, courses, objectives);**

**return 0;**

**}**

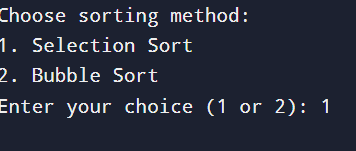
# **Screen Shots**

# **Input**

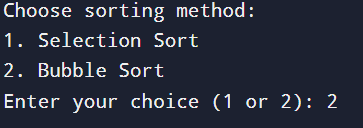


***Choose a sorting algorithm:***

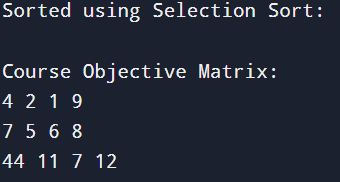
***For selection sort:***

******

***For Bubble sort:***

******

# **Sorted using selection sort :**



# **Sorted using Bubble sort :**

# 

# **CONCLUSION**

In the programming part, we developed a full C program to assist in understanding and comparing sorting algorithms more specifically, selection sort and bubble sort. The features of this program will include the possible implementation for both algorithms regarding the performance of different datasets on the same set of data, print and present sorted results, along with the detailed explanation of capturing output screenshots and documenting the code using a flowchart that shows the control flow of each sorting method.

In doing all of this, you learned through the implementation and comparison of these sorting algorithms, thereby creating value in understanding the designs of algorithms, their performance trade-offs, and their practical programming, adding impact to your problem-solving skills, making sense of all the different techniques used in sorting, and enhancing your ability to effectively put across technical information.

THANK YOU!