**OBE IMPLEMENTATION : COURSE OBJECTIVE SETTING**

by

**S.Revanth Sai[AP23110011112]**

**T. Amruthaa[AP23110011142]**

**K.Praneeth Chandra[AP23110011079]**

**K.Mokshagna[AP23110011110]**

**K.Kiranmai[AP23110010716]**

**Navaneeth[AP23110011106]**

A report for the CSE204: Design and Analysis of Algorithm project



Department of Computer science and Engineering

**SRM UNIVERSITY AP::AMARAVATI** **INDEX**

* **Introduction**
* **Module description**
* **Architecture diagram**
* **Field/table details**
* **Algorithm Details**
* **Source code**
* **Screen Shots**
* **Conclusion**

# 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.

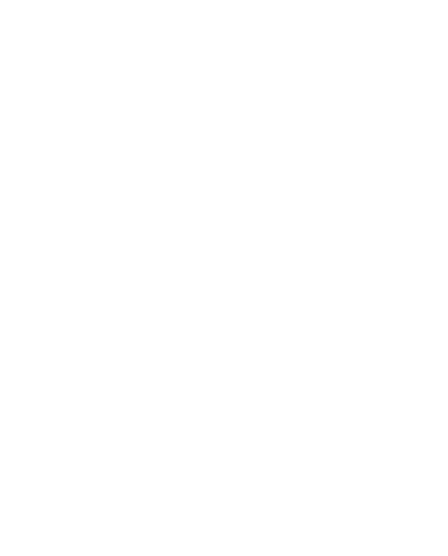
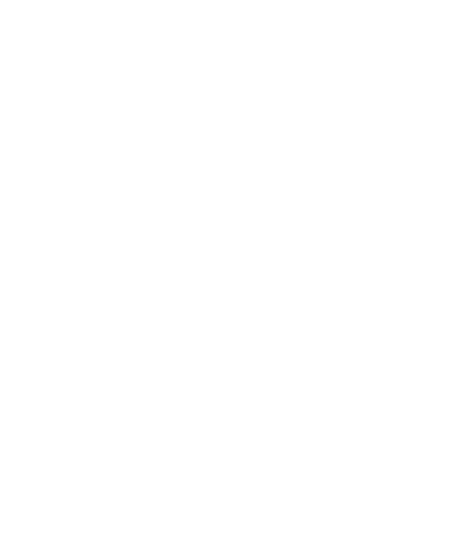
**PROJECT MODULES: Course Objective Setting**

**Architecture Diagram**

# 

# 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.**



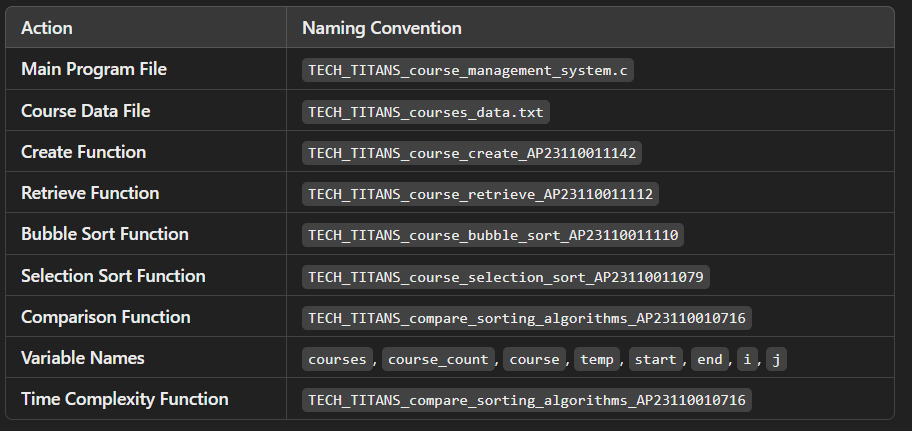
## Programming Details and Naming Conventions

* **File Naming Convention:**
* **Main Program**: TECH\_TITANS\_course\_management\_system.c
* **Data File**: TECH\_TITANS\_courses\_data.txt
* **Function Naming Convention:**

Functions should follow the pattern: **[Team Name]\_[Function Purpose]\_[Unique Identifier]**

* **Create Course**:  
  TECH\_TITANS\_course\_create\_AP23110011142
  + Adds a new course to the system.
* **Retrieve Courses**:  
  TECH\_TITANS\_course\_retrieve\_AP23110011112
  + Displays the list of all courses.
* **Bubble Sort**:  
  TECH\_TITANS\_course\_bubble\_sort\_AP23110011110
  + Sorts courses using the Bubble Sort algorithm.
* **Selection Sort**:  
  TECH\_TITANS\_course\_selection\_sort\_AP23110011079
  + Sorts courses using the Selection Sort algorithm.
* **Compare Sorting Algorithms**:  
  TECH\_TITANS\_compare\_sorting\_algorithms\_AP23110010716
  + Compares performance of Bubble Sort and Selection Sort.
* **Variable Naming:**
* **Global Variables**: courses, course\_count
* **Temporary Variables**: course, temp, start, end
* **Loop Counters**: i, j.

# 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>**

**#include <string.h>**

**#include <time.h>**

**#define MAX 100**

**typedef struct {**

**int course\_id;**

**char course\_code[10];**

**char course\_name[50];**

**char course\_description[100];**

**float course\_credits;**

**} Course;**

**Course courses[MAX];**

**int course\_count = 0;**

**void create\_course\_AP23110011142();**

**void retrieve\_courses\_AP23110011112();**

**void bubble\_sort\_courses\_AP23110011110();**

**void selection\_sort\_courses\_AP23110011079();**

**void compare\_sorting\_algorithms\_AP23110010716();**

**int main() {**

**int choice;**

**while (1) {**

**printf("\n1. Create Course\n2. Retrieve Courses\n3. Bubble Sort\n4. Selection Sort\n5. Compare Sorting\n6. Exit\n");**

**printf("Enter your choice: ");**

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

**getchar(); // Clear newline character left by scanf**

**switch (choice) {**

**case 1: create\_course\_AP23110011142(); break;**

**case 2: retrieve\_courses\_AP23110011112(); break;**

**case 3: bubble\_sort\_courses\_AP23110011110(); break;**

**case 4: selection\_sort\_courses\_AP23110011079(); break;**

**case 5: compare\_sorting\_algorithms\_AP23110010716(); break;**

**case 6: return 0;**

**default: printf("Invalid choice!\n");**

**}**

**}**

**return 0;**

**}**

**void create\_course\_AP23110011142() {**

**if (course\_count >= MAX) {**

**printf("Course list is full!\n");**

**return;**

**}**

**Course c;**

**printf("Enter Course ID: ");**

**scanf("%d", &c.course\_id);**

**getchar(); // Clear the newline character left by scanf**

**printf("Enter Course Code: ");**

**fgets(c.course\_code, sizeof(c.course\_code), stdin);**

**c.course\_code[strcspn(c.course\_code, "\n")] = '\0'; // Remove trailing newline**

**printf("Enter Course Name: ");**

**fgets(c.course\_name, sizeof(c.course\_name), stdin);**

**c.course\_name[strcspn(c.course\_name, "\n")] = '\0'; // Remove trailing newline**

**printf("Enter Course Description: ");**

**fgets(c.course\_description, sizeof(c.course\_description), stdin);**

**c.course\_description[strcspn(c.course\_description, "\n")] = '\0'; // Remove trailing newline**

**printf("Enter Course Credits: ");**

**scanf("%f", &c.course\_credits);**

**getchar(); // Clear the newline character left by scanf**

**// Store the course in the array**

**courses[course\_count++] = c;**

**printf("Course added successfully!\n");**

**}**

**void retrieve\_courses\_AP23110011112() {**

**if (course\_count == 0) {**

**printf("No courses available.\n");**

**return;**

**}**

**printf("\nCourse List:\n");**

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

**printf("ID: %d, Code: %s, Name: %s, Description: %s, Credits: %.2f\n",**

**courses[i].course\_id, courses[i].course\_code,**

**courses[i].course\_name, courses[i].course\_description,**

**courses[i].course\_credits);**

**}**

**}**

**void bubble\_sort\_courses\_AP23110011110() {**

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

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

**if (strcmp(courses[j].course\_code, courses[j + 1].course\_code) > 0) {**

**Course temp = courses[j];**

**courses[j] = courses[j + 1];**

**courses[j + 1] = temp;**

**}**

**}**

**}**

**printf("Courses sorted using Bubble Sort!\n");**

**retrieve\_courses\_AP23110011112();**

**}**

**void selection\_sort\_courses\_AP23110011079() {**

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

**int min\_idx = i;**

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

**if (strcmp(courses[j].course\_code, courses[min\_idx].course\_code) < 0) {**

**min\_idx = j;**

**}**

**}**

**Course temp = courses[min\_idx];**

**courses[min\_idx] = courses[i];**

**courses[i] = temp;**

**}**

**printf("Courses sorted using Selection Sort!\n");**

**retrieve\_courses\_AP23110011112();**

**}**

**void compare\_sorting\_algorithms\_AP23110010716() {**

**clock\_t start, end;**

**start = clock();**

**bubble\_sort\_courses\_AP23110011110();**

**end = clock();**

**printf("Bubble Sort Time: %lf seconds\n", (double)(end - start) / CLOCKS\_PER\_SEC);**

**start = clock();**

**selection\_sort\_courses\_AP23110011079();**

**end = clock();**

**printf("Selection Sort Time: %lf seconds\n", (double)(end - start) / CLOCKS\_PER\_SEC);**

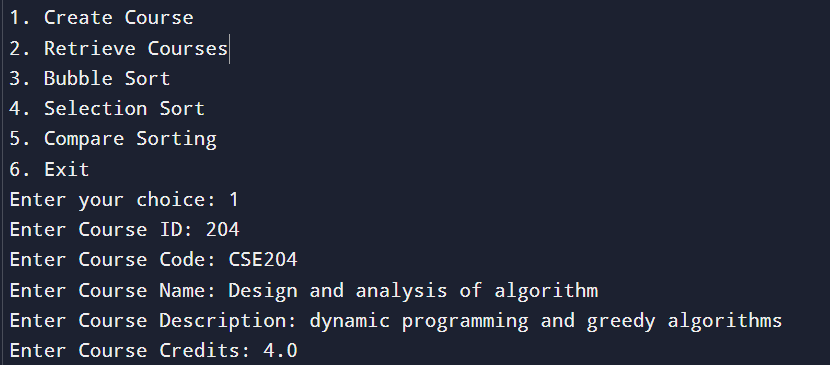
**printf("Comparison performed by AP23110011106.\n");**

**}**

# **Screen Shots**

## Create Course :

## INPUT :

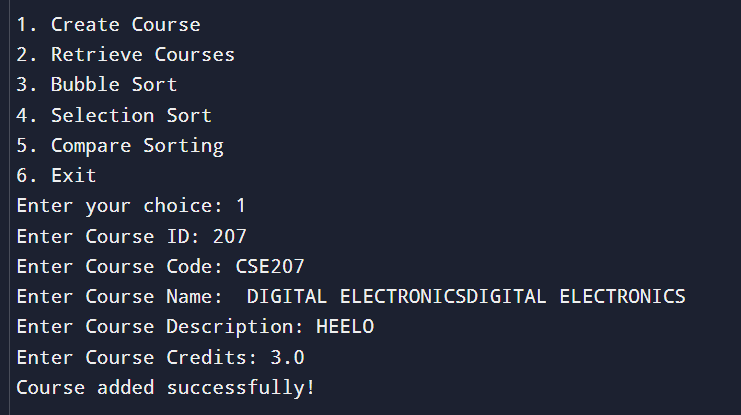


## OUTPUT :



## CREATING ANOTHER COURSE :

## INPUT :

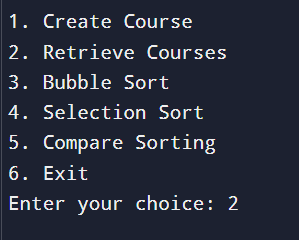


## OUTPUT :

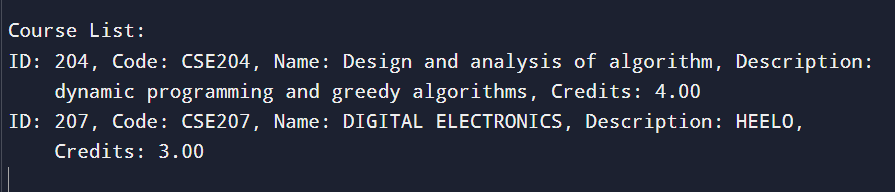


## RETRIEVE COURSES

## INPUT :

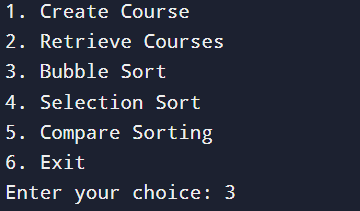


## OUTPUT :

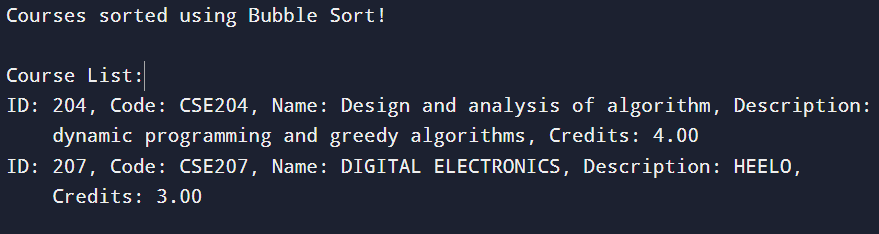


# **BUBBLE SORT COURSES :**

# **INPUT :**

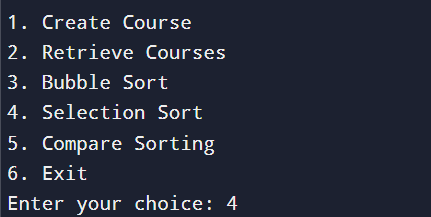


# **OUTPUT :**

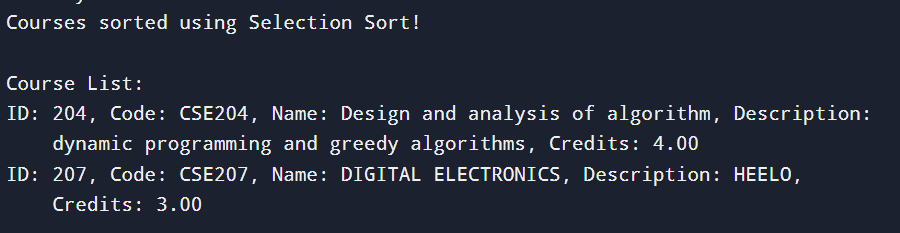


# **SELECTION SORT COURSES :**

# **INPUT:**



# **OUTPUT:**

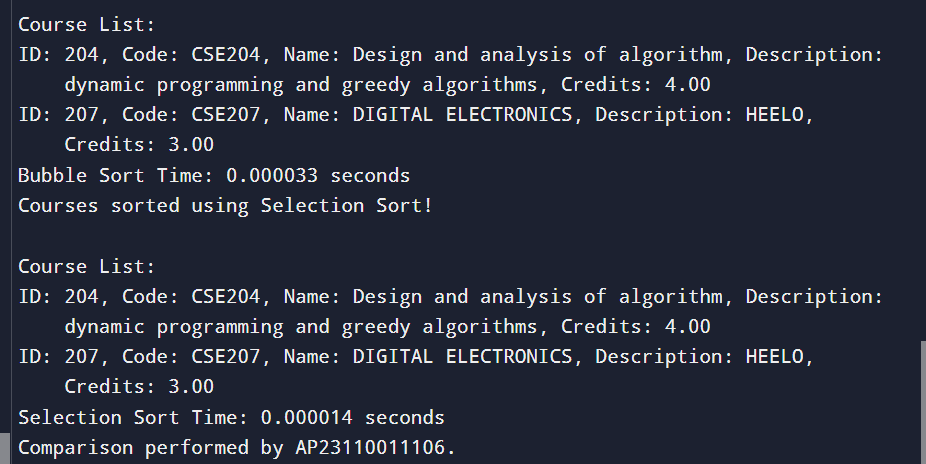


## COMPARE SORTING ALGORITHMS:

## INPUT:

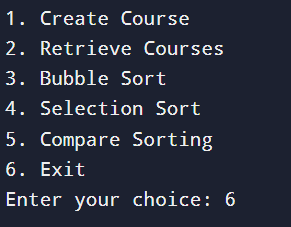
## 

## OUTPUT:



## EXIT THE PROGRAM :

## INPUT:



## OUTPUT :



# **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, we 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!