# Student Information

| Name | Section | Student Code |
|---|---|---|
| أحمد علي أحمد علي عثمان | 1 | 20240592 |

## Task 1: Store and Display Total Marks Using a 2D Array

- **Objective**: Write a C program to store marks of 5 students in 3 subjects using a 2D array, then calculate and display total marks for each student.
- **Code**:

```c
#include <stdio.h>

int main() {
  // 2D array for 5 students, 3 subjects
  int marks[5][3];
  int total; // Variable for total marks per student

  // Input marks
  for (int i = 0; i < 5; i++) { // Loop through students
    printf("\nEnter marks for student %d:\n", i + 1);

    for (int j = 0; j < 3; j++) { // Loop through subjects
      printf("Subject %d: ", j + 1);
      scanf("%d", &marks[i][j]); // Read marks
    }
  }

  printf("\n"); // Newline for output formatting

  // Calculate and display total marks
  for (int i = 0; i < 5; i++) { // Loop through students
    total = 0; // Reset total for each student

    for (int j = 0; j < 3; j++) { // Loop through subjects to sum marks
      total += marks[i][j]; // Add subject mark to total
    }

    // Display total marks for the current student
    printf("Total marks for student %d: %d\n", i + 1, total);
  }

  return 0; // Indicate success
}
```

## Task 2: Create and Use a Student Structure

- **Objective**: Define a `student` structure with ID, name, and three subject marks, then input and display data for 3 students.
- **Code**:

```c
#include <stdio.h>

// Define a structure named Student
struct Student {
  int id; // Student ID
  char name[50]; // Student name (character array to hold the string)
  int marks[3]; // Array to hold marks for 3 subjects
};

int main() {
  // Declare an array of 3 Student structures
  struct Student students[3];

  // Input data for 3 students
  for (int i = 0; i < 3; i++) { // Loop through each student
    printf("Enter details for student %d:\n", i + 1); // Prompt for student details
    printf("ID: ");
    scanf("%d", &students[i].id); // Read student ID

    printf("Name: ");
    // Read student name. scanf("%s", ...) reads a single word and stops at whitespace.
    // Be cautious with scanf("%s") for names with spaces; fgets is generally safer.
    scanf("%s", students[i].name);

    // Input marks for the 3 subjects for the current student
    for (int j = 0; j < 3; j++) { // Loop through each subject
      printf("Subject %d marks: ", j + 1);
      scanf("%d", &students[i].marks[j]); // Read subject marks
    }
  }

  printf("--------------------\n"); // Separator line for output

  // Display data for the 3 students
  for (int i = 0; i < 3; i++) { // Loop through each student
    printf("\nStudent %d:\n", i + 1); // Display student number
    printf("ID: %d\n", students[i].id); // Display student ID
    printf("Name: %s\n", students[i].name); // Display student name

    // Display marks for the 3 subjects for the current student
    for (int j = 0; j < 3; j++) { // Loop through each subject
      printf("Subject %d marks: %d\n", j + 1, students[i].marks[j]); // Display subject marks
    }
  }

  return 0;
}
```

## Task 3: Calculate Average with a Function

- **Objective:** Write a `calculateAverage()` function that takes a `student` structure, calculates the average of marks, and updates the structure.
- **Code:**

C

```c
#include <stdio.h>

// Define a structure named Student
struct Student {
  int id; // Student ID
  char name[50]; // Student name
  int marks[3]; // Array to hold marks for 3 subjects
  float average; // Variable to store the calculated average
};

// Function to calculate the average marks for a student
// Takes a pointer to a Student structure as input
void calculateAverage(struct Student *s) {
  float sum = 0; // Initialize sum of marks

  // Loop through the marks array to calculate the sum
  for (int i = 0; i < 3; i++) {
    sum += s->marks[i]; // Add current subject's mark to sum (using pointer dereference ->)
  }

  // Calculate the average and store it in the average field of the structure
  s->average = sum / 3.0; // Use 3.0 for floating-point division
}

int main() {
  // Declare and initialize a Student structure variable 's1'
  struct Student s1 = {
    .id = 1, // Initialize ID
    .name = "John", // Initialize name
    .marks = { 80, 85, 90 }, // Initialize marks for 3 subjects
    .average = 0.0 // Initialize average (will be calculated later)
  };

  // Display student details before calculating the average
  printf("Before\n");
  printf("Student ID: %d\nName: %s\nAverage: %.2f\n", s1.id, s1.name, s1.average);

  // Call the calculateAverage function, passing the address of s1
  // This allows the function to modify the s1 structure directly
  calculateAverage(&s1);

  // Display student details after calculating the average
  printf("\nAfter\n");
  printf("Student ID: %d\nName: %s\nAverage: %.2f\n", s1.id, s1.name, s1.average);

  return 0;
}
```

## Task 4: Use Pointers to Modify Student ID

- **Objective**: Declare a student ID variable and use pointers to modify and display it.
- **Code**:

```c
#include <stdio.h>

```

```c
int main() {
  // Declare an integer variable and initialize it
  int studentId = 100;

  // Declare an integer pointer and initialize it to point to the memory address of studentId
  int* idPtr = &studentId;

  // Print the initial value of studentId using the variable name
  printf("student ID before edit: %d\n", studentId);

  // Change the value of the variable studentId using the pointer
  // The '*' operator dereferences the pointer, accessing the value at the memory address it points to
  *idPtr = 101;

  // Print the updated value of studentId using the variable name
  printf("student ID after edit (var): %d\n", studentId);

  // Print the updated value of studentId using the pointer dereference
  // This shows that the value pointed to by idPtr has also changed
  printf("student ID after edit (pointer): %d\n", *idPtr);

  return 0;
}
```

## Task 5: Compare Arrays and Structures

- **Arrays**
  - Hold multiple elements of the *same* type (e.g. all `int` or all `double`).
  - Laid out contiguously in memory—you can index into them (`arr[5]`) and perform pointer-arithmetic.
  - Good when you need a simple list or table of values, all governed by the same operations.
- **Structures**
  - Bundle together one or more variables, possibly of *different* types, under a single name.
  - Each member can be accessed by name (`person.age`, `person.name`), improving clarity when you have logically related but type-varying data.
  - Memory layout may include padding to satisfy alignment, but you don't lose the benefit of grouping.

> **When to prefer a `struct` over separate arrays?**
> Use a `struct` when you're modelling an entity that has multiple attributes—especially if those attributes are not all the same data type. With arrays you'd need parallel arrays (e.g. `ages[i]`, `names[i]`, `grades[i]`), which is error-prone and scatters related data. A `struct` keeps each "record" intact, makes function interfaces cleaner, and aligns with best practices for data encapsulation.

```c
#include <stdio.h>

/* Define a student record with mixed fields */
struct Student {
  char name[32];
  int age;
  float gpa;
};

int main() {
  /* Array of structures: each element is a complete student record */
  struct Student classroom[3] = {
    { "Alice", 20, 3.8f },
    { "Bob", 22, 3.2f },
    { "Cara", 19, 3.9f }
  };

  /* Print each student’s data */
  for (int i = 0; i < 3; i++) {
    printf(
      "Student %d: %s, age %d, GPA %.2f\n",
      i + 1,
      classroom[i].name,
      classroom[i].age,
      classroom[i].gpa
    );
  }

  return 0;
}
```

- **Why this is better than parallel arrays**
  - You avoid mistakes like shifting one array and not the others.
  - Passing one `Student` to a function is simpler than passing three separate arrays plus an index.
  - The code is self-documenting: `student.age` vs. `ages[i]`.

# Task 6: Dynamic Access with Pointers

- **Objective**: Store marks of 5 students in a 1D array and use pointers to display them.
- **Code**:

```c
#include <stdio.h>

int main() {
  // Declare a one-dimensional array to store marks for 5 students in 3 subjects.
  // Total elements = 5 students * 3 subjects = 15.
  int marks[15];

  // Input marks for 5 students (3 subjects each) into the 1D array.
  // The index calculation i * 3 + j maps the 2D logic (student i, subject j)
  // to the 1D array index.
  for (int i = 0; i < 5; i++) { // Loop through each student
    printf("Enter marks for student %d:\n", i + 1);

    for (int j = 0; j < 3; j++) { // Loop through each subject
      printf("Subject %d: ", j + 1);
      scanf("%d", &marks[i * 3 + j]); // Read marks into the calculated 1D index
    }
  }

  // Declare an integer pointer and initialize it to point to the beginning of the 'marks' array.
  // The array name 'marks' itself acts as a pointer to its first element.
  int* ptr = marks;

  printf("---------------------\n"); // Separator line for output


  // Loop to display marks using the pointer to traverse the array.
  for (int i = 0; i < 5; i++) { // Loop through each student
    printf("Student %d marks: ", i + 1);

    for (int j = 0; j < 3; j++) { // Loop through each subject for the current student
      printf("%d ", *ptr); // Dereference the pointer to access the value at the current memory location
      ptr++; // Increment the pointer to move to the next integer element in the array
    }

    printf("\n"); // Newline after displaying marks for each student
  }


  return 0;
}
```

## Task 7: Complete Student Grading System

- **Objective**: Build a system using arrays, structures, functions, and pointers to input, calculate averages, and display a report.
- **Code**:

```c
#include <stdio.h>
#include <string.h> // Include string library for string manipulation functions

// Define a structure named Student
struct Student {
  int id; // Student ID
  char name[50]; // Student name
  int marks[3]; // Array to hold marks for 3 subjects
  float average; // Variable to store the calculated average
};

// Function to calculate the average marks for a student
// Takes a pointer to a Student structure as input
void calculateAverage(struct Student *s) {
  float sum = 0; // Initialize sum of marks

  // Loop through the marks array to calculate the sum
  for (int i = 0; i < 3; i++) {
    sum += s->marks[i]; // Add current subject's mark to sum (using pointer dereference ->)
  }

  // Calculate the average and store it in the average field of the structure
  s->average = sum / 3.0; // Use 3.0 for floating-point division
}
```

```c

// Function to read a line from input, removing the newline character
void readline(char* restrict s, int n, FILE *restrict stream) {
  fgets(s, n, stream);
  s[strcspn(s, "\n")] = '\0'; // Remove the newline '\n' character
}

// Function to clear the input buffer
void clearInputBuffer() {
  int c;

  while ((c = getchar()) != '\n' && c != EOF);
}

int main() {
  // Declare an array of 5 Student structures
  struct Student students[5];

  // Input data for 5 students
  for (int i = 0; i < 5; i++) { // Loop through each student
    printf("Enter details for student %d:\n", i + 1); // Prompt for student details

    printf("ID: ");
    scanf("%d", &students[i].id); // Read student ID
    clearInputBuffer(); // Clear buffer after reading integer

    printf("Name: ");
    readline(students[i].name, 50, stdin); // Read student name (handles spaces)

    // Input marks for the 3 subjects for the current student
    for (int j = 0; j < 3; j++) { // Loop through each subject
      printf("Subject %d marks: ", j + 1);
      scanf("%d", &students[i].marks[j]); // Read subject marks
    }

    calculateAverage(&students[i]); // Calculate average for the student
  }

  printf("\nStudent Grading Report:\n"); // Header for the report

  // Display data for the 5 students
  for (int i = 0; i < 5; i++) { // Loop through each student
    struct Student s = students[i]; // Create a copy for easier access (optional, could use students[i] directly)
    printf(
      "ID: %d, Name: '%s', Marks: %d, %d, %d, Average: %.2f\n",
      s.id, s.name, s.marks[0], s.marks[1], s.marks[2], s.average // Print student details and calculated average
    );
  }

  // Demonstrate updating a student's mark using a pointer
  struct Student *p = &students[0]; // Declare a pointer 'p' and point it to the first student structure

  p->marks[0] = 90; // Update the first subject's mark for the first student using the pointer

  calculateAverage(p); // Recalculate the average for the first student after updating marks

  printf("\nAfter updating student 1's subject 1 mark to 90:\n"); // Message indicating the update
  printf(
    "ID: %d, Name: '%s', Marks: %d, %d, %d, Average: %.2f\n",
    p->id, p->name, p->marks[0], p->marks[1], p->marks[2], p->average // Display updated details
  );

  return 0;
}
```

## Task 8: Save Names and Averages to File

- **Objective**: Save student names and averages to `grades.txt`.
- **Code**:

```c
#include <stdio.h>
#include <string.h> // String manipulation functions

// Define filename as a constant
const char FILENAME[11] = "grades.txt";

// Student structure
struct Student {
  char name[50]; // Student name
  int marks[3]; // Marks for 3 subjects
  float average; // Calculated average
};

// Function to calculate average marks
void calculateAverage(struct Student *s) {
  float sum = 0;

  for (int i = 0; i < 3; i++) {
    sum += s->marks[i];
  }

  s->average = sum / 3.0;
}

// Function to read a line, removing newline
void readline(char* restrict s, int n, FILE *restrict stream) {
  fgets(s, n, stream);
  s[strcspn(s, "\n")] = '\0';
}

// Function to clear input buffer
void clearInputBuffer() {
  int c;

  while ((c = getchar()) != '\n' && c != EOF);
}

```

```c
int main() {
  // Open file in write mode ("w"). Creates or overwrites.
  FILE *file = fopen(FILENAME, "w");
  int character; // Variable for reading characters

  // Check if file opened successfully for writing
  if (file == NULL) {
    printf("Error opening %s!\n", FILENAME);
    return 1;
  }

  // Array of 3 Student structures
  struct Student students[3];

  // Input data for 3 students
  for (int i = 0; i < 3; i++) {
    printf("Enter details for student %d:\n", i + 1);
    printf("Name: ");
    readline(students[i].name, 50, stdin);

    for (int j = 0; j < 3; j++) {
      printf("Subject %d marks: ", j + 1);
      scanf("%d", &students[i].marks[j]);
    }

    clearInputBuffer(); // Clear buffer after scanf

    calculateAverage(&students[i]); // Calculate average
  }

  // Write student name and average to file
  for (int i = 0; i < 3; i++) {
    struct Student s = students[i];
    fprintf(file,
      "Name: '%s', Average: %.2f\n",
      s.name, s.average
    );
  }

  // Close the file after writing
  fclose(file);

  // Reopen the file in read mode ("r")
  file = fopen(FILENAME, "r");

  // Check if file opened successfully for reading
  if (file == NULL) {
    printf("Error reading from %s!\n", FILENAME);
    return 1;
  }

  printf("Reading content from %s:\n\n", FILENAME);

  // Read and print file content character by character
  while ((character = fgetc(file)) != EOF) {
    putchar(character);
  }

  // Close the file after reading
  fclose(file);

  return 0; // Indicate success
}
```

## Task 9: Explain "w" vs. "a" File Modes

| Mode | Behavior | Existing file content |
|---|---|---|
| `"w"` | Open for writing. If the file exists, truncate it to zero length (erase all data). If absent, create new file. | Discarded (file is cleared) |
| `"a"` | Open for writing. If the file exists, writing always goes to the end (append). If absent, create new file. | Preserved; new data added at end |

> **Key point:**
>
> - Use `"w"` when you want to start fresh and do not care about any previous content.
> - Use `"a"` when you want to preserve existing content and add more data at the end.

- **Code**:

```c
#include <stdio.h>

int main() {
  FILE *fp;

  // Write mode
  fp = fopen("test.txt", "w");
  fprintf(fp, "This is write mode.\n");
  fclose(fp);

  // Append mode
  fp = fopen("test.txt", "a");
  fprintf(fp, "This is append mode.\n");
  fclose(fp);

  // Read and display
  fp = fopen("test.txt", "r");
  char line[100];
  while(fgets(line, 100, fp) != NULL) {
    printf("%s", line);
  }
  fclose(fp);
  return 0;
}
```

- **Output Explanation**: First write creates/overwrites with "This is write mode." Append adds "This is append mode." Result: both lines in the file.

## Task 10: Read and Display Records from File

- **Objective**: Read student records from `grades.txt` and display them.
- **Code**:

```c
#include <stdio.h> // Standard I/O library
#include <string.h> // String manipulation functions

// Define filename
const char FILENAME[11] = "grades.txt";

int main() {
  // Open file in read mode ("r")
  FILE *file = fopen(FILENAME, "r");

  // Check for file open errors
  if (file == NULL) {
    printf("Error opening file.\n");
    return 1; // Indicate error
  }

  char line[100]; // Buffer for reading lines
  printf("Student Grades Report:\n"); // Report header

  // Read file line by line using fgets
  while (fgets(line, 100, file) != NULL) {
    char name[50]; // Variable for name
    float average; // Variable for average

    /*
      Parse line using sscanf:
      - "Name: %[^,]" extracts name up to comma.
      - ", " matches literal comma and space.
      - "Average: %f" extracts float after "Average: ".
    */
    sscanf(line, "Name: %[^,], Average: %f", name, &average);

    // Print extracted data
    printf("Name: %s, Average: %.2f\n", name, average);
  }

  // Close the file
  fclose(file);

  return 0; // Indicate success
}
```

## Task 11: Complete Grading Application with File Handling

- **Objective**: Design an application to load data, update marks, recalculate averages, and save back to a file.
- **Code**:

```c
#include <stdio.h> // Standard I/O library for file operations, printf, scanf
#include <stdlib.h> // Standard library for exit()
#include <string.h> // String manipulation functions for strcspn, strcpy

// Student structure definition
struct Student {
  int id; // Student ID
  char name[50]; // Student name
  int marks[3]; // Marks for 3 subjects
  float average; // Calculated average
};

// Function to calculate average marks for a student
void calculateAverage(struct Student *s) {
  float sum = 0;

  for (int i = 0; i < 3; i++) {
    sum += s->marks[i];
  }

  s->average = sum / 3.0;
}

// Function to read a line from input, removing newline
void readline(char* restrict s, int n, FILE *restrict stream) {
  fgets(s, n, stream);
  s[strcspn(s, "\n")] = '\0'; // Remove the newline '\n' character
}

// Function to clear input buffer after scanf
void clearInputBuffer() {
  int c;

  while ((c = getchar()) != '\n' && c != EOF);
}

// Function to read student details from user input
void readStudent(struct Student *s) {
  printf("\nEnter student ID: ");
  scanf("%d", &s->id);
  clearInputBuffer(); // Clear buffer after reading integer

  printf("Name: ");
  readline(s->name, 50, stdin); // Read name (handles spaces)

  for (int j = 0; j < 3; j++) {
    printf("Subject %d marks: ", j + 1);
    scanf("%d", &s->marks[j]); // Read subject marks
  }

  clearInputBuffer(); // Clear buffer after reading marks

  calculateAverage(s); // Calculate average after input
}


// Function to create an empty file
// Returns 0 on success, 1 on failure
int createFile(const char* filename) {
  FILE *file;
  file = fopen(filename, "w"); // Open file in write mode (creates or clears)

  if (file == NULL) {
    printf("Error: Could not create file %s\n", filename);
    return 1; // Indicate error
  }

  fclose(file); // Close immediately to leave it empty
  return 0; // Indicate success
}

// Function to save student data to a file
void saveData(struct Student students[], int n, const char* filename) {
  FILE *file = fopen(filename, "w"); // Open file in write mode

  if (file == NULL) {
    printf("Error opening file for saving.\n");
    exit(1); // Exit on critical error
  }

  // Write student data in a formatted way
  for (int i = 0; i < n; i++) {
    fprintf(
      file, "%d %s %d %d %d\n", // Format: ID Name Mark1 Mark2 Mark3
      students[i].id, students[i].name, students[i].marks[0],
      students[i].marks[1], students[i].marks[2]
    );
  }

  fclose(file); // Close file after saving
}


// Function to load student data from a file
// 'checked' flag prevents infinite recursion on file not found
void loadData(struct Student students[], int n, const char* filename, short checked) {
  FILE *fp = fopen(filename, "r"); // Open file in read mode

  if (fp == NULL) {
    if (checked) {
      printf("Error opening file for loading.\n");
      exit(1); // Exit if file should exist but doesn't
    } else {
      // If file doesn't exist on first try, create it, get input, save, and try loading again
      printf("File not found. Creating new file and getting student data.\n");
      createFile(filename);

      for (int i = 0; i < n; i++) {
        readStudent(&students[i]); // Get student data from user
      }

      saveData(students, n, filename); // Save the newly entered data

      loadData(students, n, filename, 1); // Try loading again (checked is now 1)
      return;
    }
  }

  // Read student data from file using fscanf
  for (int i = 0; i < n; i++) {
    fscanf(
      fp, "%d %s %d %d %d", // Format to match saveData
```

```c
      &students[i].id, students[i].name, &students[i].marks[0],
      &students[i].marks[1], &students[i].marks[2]
    );
    calculateAverage(&students[i]); // Calculate average after loading marks
  }

  fclose(fp); // Close file after loading
}

int main() {
  struct Student students[5]; // Array to hold 5 students
  const char* filename = "students.txt"; // Data file name

  // Load student data from file (or get input if file doesn't exist)
  loadData(students, 5, filename, 0);

  // Display current student data
  printf("\nCurrent Student Data:\n");

  for (int i = 0; i < 5; i++) {
    printf(
      "ID: %d, Name: %s, Marks: %d, %d, %d, Average: %.2f\n",
      students[i].id, students[i].name, students[i].marks[0],
      students[i].marks[1], students[i].marks[2], students[i].average
    );
  }

  // Update marks section
  int id;
  printf("\nEnter student ID to update: ");
  scanf("%d", &id); // Get ID to update
  int index = -1; // Index of student to update

  // Find the student by ID
  for (int i = 0; i < 5; i++) {
    if (students[i].id == id) {
      index = i; // Found student, store index
      break;
    }
  }

  // Handle student not found
  if (index == -1) {
    printf("Student not found.\n");
    return 1; // Exit or handle error
  }

  // Get new marks for the selected student
  printf("Enter new marks for student %d:\n", id);

  for (int j = 0; j < 3; j++) {
    printf("Subject %d: ", j + 1);
    scanf("%d", &students[index].marks[j]); // Read new marks
  }

  calculateAverage(&students[index]); // Recalculate average after update

  // Display updated student data
  printf("\nUpdated Student Data:\n");

  for (int i = 0; i < 5; i++) {
    printf(
      "ID: %d, Name: %s, Marks: %d, %d, %d, Average: %.2f\n",
      students[i].id, students[i].name, students[i].marks[0],
      students[i].marks[1], students[i].marks[2], students[i].average
    );
  }

  // Save the updated data back to the file
  saveData(students, 5, filename);
  printf("\nData saved to %s\n", filename);

  return 0; // Indicate successful execution
}
```