# Working with Structs and Pointers to Structs

## Lab Assignment: Working with Structs and Pointers to Structs

### Objectives

- Understand how to define and initialize structs in C.
- Practice using pointers to structs and accessing struct members through pointers.
- Learn to pass structs and pointers to functions.
- Gain experience dynamically allocating memory for structs using `malloc` and releasing it with `free`.

### Instructions

You will create a simple program that manages information about students enrolled in a class. Each student will have attributes such as name, ID, and grades. Follow the steps below to complete the lab.

## Part 1: Define the Struct

1. Create a struct named `Student` that has the following members:
    - `char name[50]`: the student's name.
    - `int id`: the student's ID number.
    - `float grade1, grade2, grade3`: the student's grades for three assignments.
2. Write a function called `init_student` that takes a pointer to a `Student` struct as a parameter, along with the student's name, ID, and three grades. This function should:
    - Use `strcpy` to set the student's name.

- Set the ID and grade values for the struct.

**Function Prototype**:

```
void init_student(struct Student *s, const char *name, int id, float g1, float g2, float g3);
```

## Part 2: Calculate Average Grade

3. Write a function called `calculate_average` that takes a pointer to a `Student` struct and returns the average of the three grades.

   **Function Prototype**:

   ```
   float calculate_average(const struct Student *s);
   ```

## Part 3: Print Student Information

4. Write a function called `print_student` that takes a pointer to a `Student` struct and prints the student's name, ID, and average grade.

   **Function Prototype**:

   ```
   void print_student(const struct Student *s);
   ```

## Part 4 (Optional): Create and Manipulate an Array of Structs

5. In your `main` function:
   - Optionally, you can create an array of `Student` structs to hold up to 5 students and use `init_student` to store data for each.
   - This step is **only for practice** and does not need to be submitted.

- If you choose to complete this part, simply verify that the functions `calculate_average` and `print_student` work as expected on the array elements.
- **Note**: This step is a **cursory check** and is not required for submission. It's an opportunity to reinforce your understanding of using structs in an array context.

## Part 5: Dynamic Allocation of a Struct Array

6. Modify your `main` function to dynamically allocate an array of `Student` structs based on the number of students the user wants to enter. To do this:
   - Ask the user how many students they wish to enter.
   - Use `malloc` to create an array of `Student` structs of the specified size.
   - After storing the data for each student, free the dynamically allocated memory.

## Sample Output

```
Enter the number of students: 2

Enter information for student 1:
Name: Alice
ID: 1001
Grade 1: 85.5
Grade 2: 90.0
Grade 3: 88.5

Enter information for student 2:
Name: Bob
ID: 1002
Grade 1: 78.0
Grade 2: 82.5
Grade 3: 80.0
```

```
Student Information:
--------------------
Name: Alice
ID: 1001
Average Grade: 88.0

Name: Bob
ID: 1002
Average Grade: 80.2
```

## Submission Instructions

Submit your `lab_structs.c` source code file once you have completed the lab. Ensure the code compiles and runs correctly before submitting.