DATE: LAB REPORT NO.: **7** SET: **A**

TITLE OF THE PROGRAM: **STRUCTURE**

BASIC STRUCTURE

OBJECTIVES

I. To understand and implement the static and dynamic structure.
II. To understand and implement array of structure.

REQUIREMENTS

1. C Compiler (e.g., GCC)
2. Computer System
3. IDE or Text Editor
4. OS compatible with the software

THEORY

Definition

Structure is a user-defined data type in C programming language that combines logically related data items of different data types together.

Syntax

```
struct structureName
{
    data_type member1;
    data_type memberN;
};
```

Declaring structure variable

You can declare structure variables in two ways:-

By struct keyword within main() function

```
struct structureName
{
    type element1;
    type element2;
};
int main()
{
    //declaring variables of structure structureName
    struct structureName structure_variable;
    return 0;
}
```

By declaring variables at the time of defining structure.

```
struct structureName
{
    type element1;
    type element2;
}variable1, variable2, ...;
```

Accessing Structure Members

Structure members can be accessed and assigned values in a number of ways. In order to assign a value to any structure member, the member name must be linked with the structure variable using a dot **.** operator also called period or member access operator.

PROCEDURE (Program Code, Comment, and Output)

1. **Static Structure Demo Program**

Program Code:

```
#include <stdio.h>
#include <string.h>

// Structure Definition
struct Person
{
    int age;
    char name[25];
    char branch[10];
    char gender; // F for female and M for male
};

int main()
{
    struct Person p1; // Declare a struct variable p1 of type Person

    //Assign value to the member of structure of different datatype
    p1.age = 81;
    strcpy(p1.name, "Muni Bahadur Shakya");
    strcpy(p1.branch, "Computer");
    p1.gender = 'M';

    // Display the information stored in the members of structure
    printf("Name\t: %s\n", p1.name);
    printf("Age\t: %d\n", p1.age);
    printf("Branch\t: %s\n", p1.branch);
    printf("Gender\t: %c\n", p1.gender);

    return 0;
}
```

Output:

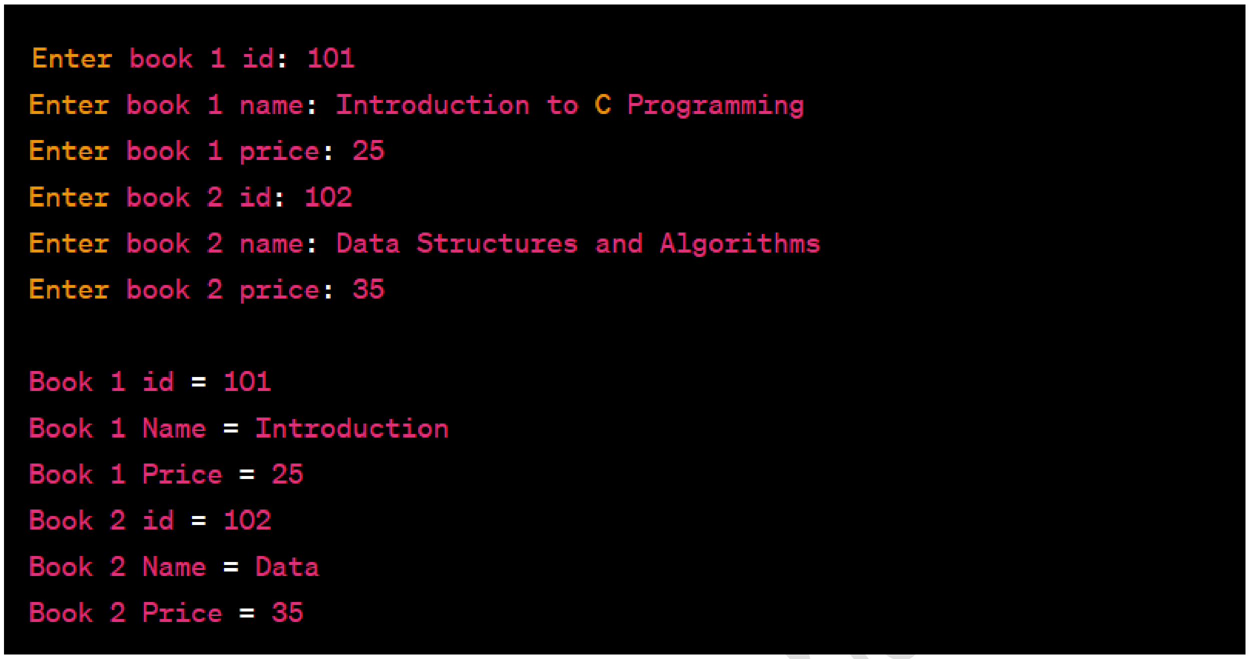
```
Name     : Muni Bahadur Shakya
Age      : 81
Branch   : Computer
Gender   : M
```

Explanation:
1. `#include <stdio.h>` and `#include <string.h>`: These are preprocessor directives that include necessary header files in the program. `<stdio.h>` provides input/output functionalities, and `<string.h>` provides string manipulation functions.
2. `struct Person`: This is a structure definition. It defines a custom data type called `Person` that contains four members: `age`, `name`, `branch`, and `gender`. Each of these members has its own data type (integer and character arrays).
3. `int main()`: This is the main function where the program execution begins.
4. `struct Person p1;`: This line declares a variable `p1` of type `Person`. This variable is a struct variable that can store information about a person, such as age, name, branch, and gender.
5. Member Assignment: The following lines of code assign values to the members of the `p1` structure variable:
  - `p1.age = 81;`: The age member of `p1` is assigned the value `81`.
  - `strcpy(p1.name, "Muni Bahadur Shakya");`: The `strcpy` function is used to copy the string "Muni Bahadur Shakya" into the `name` member of `p1`.
  - `strcpy(p1.branch, "Computer");`: The `strcpy` function is used to copy the string "Computer" into the `branch` member of `p1`.
  - `p1.gender = 'M';`: The `gender` member of `p1` is assigned the character value `'M'`.
6. Display Information: The following lines of code use `printf` to display the information stored in the members of the `p1` structure:
  - `printf("Name\t: %s\n", p1.name);`: Print the name stored in the `name` member of `p1`.
  - `printf("Age\t: %d\n", p1.age);`: Print the age stored in the `age` member of `p1`.
  - `printf("Branch\t: %s\n", p1.branch);`: Print the branch stored in the `branch` member of `p1`.
  - `printf("Gender\t: %c\n", p1.gender);`: Print the gender stored in the `gender` member of `p1`.
7. `return 0;`: This line indicates the end of the `main` function. The value `0` is returned to the operating system, indicating that the program has been executed successfully.

2. Dynamic Structure Demo Program

Program Code:
```
#include <stdio.h>
#include <string.h>

struct Book
{
    int id, price;
    char name[100];
} b1, b2;

int main()
{
    printf("Enter book 1 id: ");
    scanf("%d", &b1.id);

    printf("Enter book 1 name: ");
    scanf("%s", &b1.name);

    printf("Enter book 1 price: ");
    scanf("%d", &b1.price);

    printf("Enter book 2 id: ");
    scanf("%d", &b2.id);

    printf("Enter book 2 name: ");
    scanf("%s", &b2.name);

    printf("Enter book 2 price: ");
    scanf("%d", &b2.price);

    printf("Book 1 id = %d\n", b1.id);
    printf("Book 1 Name = %s\n", b1.name);
    printf("Book 1 Price = %d\n", b1.price);

    printf("Book 2 id = %d\n", b2.id);
    printf("Book 2 Name = %s\n", b2.name);
    printf("Book 2 Price = %d\n", b2.price);

    return 0;
}
```

Output:



```
Enter book 1 id: 101
Enter book 1 name: Introduction to C Programming
Enter book 1 price: 25
Enter book 2 id: 102
Enter book 2 name: Data Structures and Algorithms
Enter book 2 price: 35

Book 1 id = 101
Book 1 Name = Introduction
Book 1 Price = 25
Book 2 id = 102
Book 2 Name = Data
Book 2 Price = 35
```

Explanation:
1. These are preprocessor directives that include the necessary header files. `stdio.h` is included to use standard input-output functions, and `string.h` is included to work with strings and character arrays.
2. The `struct Book` defines a new user-defined data type `Book`, which contains three members: `id` (integer), `price` (integer), and `name` (character array of size 100).
3. Two variables `b1` and `b2` of type `struct Book` are declared. These variables will be used to store information about two books.
4. This is the entry point of the program. The `main` function returns an integer value (0 in this case) to the operating system upon successful execution.
5. The program prompts the user to enter information for the first book (`b1`) and the second book (`b2`) using `printf` and reads the input using `scanf`.
6. After taking the input, the program prints the information about both books using `printf` statements.
7. The program prompts the user to enter book information, and after providing the required input, it displays the entered information for both books.
8. The code read an entire line of input with spaces.

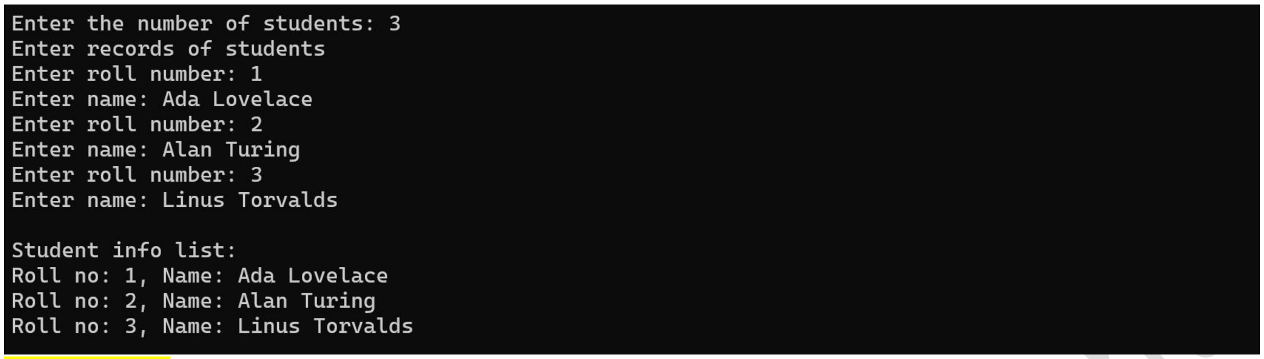
3. Array of Structure Demo Program

Program Code:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Define a structure to hold student details
struct student
```

```
{
    int rollno;
    char name[20];
};

int main()
{
    int i, size;

    // Get the number of students (size) from the user
    printf("Enter the number of students: ");
    scanf("%d", &size);

    // Create an array of 'size' student structures
    struct student st[size];

    // Prompt the user to enter records of students
    printf("Enter records of students\n");
    for (i = 0; i < size; i++)
    {
        // Read the roll number of the student
        printf("Enter roll number: ");
        scanf("%d", &st[i].rollno);

        fflush(stdin);
        // Read the name of the student (up to 9 characters, leave 1
space for the null terminator)
        printf("Enter name: ");
        gets(st[i].name);
    }

    // Display the information of all students entered
    printf("\nStudent info list:\n");
    for (i = 0; i < size; i++)
    {
        printf("Roll no: %d, Name: %s\n", st[i].rollno, st[i].name);
    }

    return 0;
}
```

Output:



```
Enter the number of students: 3
Enter records of students
Enter roll number: 1
Enter name: Ada Lovelace
Enter roll number: 2
Enter name: Alan Turing
Enter roll number: 3
Enter name: Linus Torvalds

Student info list:
Roll no: 1, Name: Ada Lovelace
Roll no: 2, Name: Alan Turing
Roll no: 3, Name: Linus Torvalds
```

Explanation:

1. The program includes the necessary header files `stdio.h` and `string.h`. The `stdio.h` file is required for standard input and output functions, while `string.h` is needed for string manipulation functions.
2. A structure named `struct student` is defined to hold the details of each student. It contains two fields: `rollno` (an integer) and `name` (a character array of size 20).
3. The `main()` function is the entry point of the program.
4. It declares variables `i` and `size`. `i` is used as a loop variable, and `size` will store the number of students to be entered.
5. The user is prompted to input the number of students they want to enter using the `printf()` and `scanf()` functions.
6. An array of `struct student` called `st` is created with the size specified by the user.
7. The program prompts the user to enter the records of students using a loop that runs `size` times. In each iteration, the user is asked to input the roll number and the name of the student.
8. The program reads the roll number using `scanf()` and stores it in the `rollno` field of the corresponding student structure.
9. The `fflush(stdin)` function is used to flush the input buffer after reading the roll number. This is done to avoid potential issues when reading strings later on.
10. The program reads the name of the student using `gets()` function. Note that using `gets()` is generally unsafe as it doesn't limit the input size, which can lead to buffer overflow. Instead, `fgets()` should be used with care to avoid this issue.
11. The loop continues until all the student details are entered and stored in the array of structures.
12. After gathering all the student information, the program prints the list of students using another loop that iterates through the array of structures and prints each student's roll number and name using `printf()`.
13. The program ends by returning 0 from the `main()` function.

CONCLUSION

From above showcases the implementation of static and dynamic structures in C programming, covering their definition, declaration, and member access. The practical examples demonstrate the use of static and dynamic structures, as well as an array of structures, highlighting the benefits of user-defined data types for organizing related data items in C programs.