1. **Define a structure in C. Explain the different types of structure declarations with examples.**
   - A structure is a collection of variables referenced under one name, providing a convenient means of keeping related information together.
   - A structure declaration forms a template that can be used to create structure objects (that is, instances of a structure).
   - The variables that make up the structure are called members. (Structure members are also commonly referred to as elements or fields.)
   - The members of a structure are logically related.
   - **The general form of a structure declaration is:**

```
struct tag {
   type member-name;
   type member-name;
   type member-name;
   .
   .
   .
} structure-variables;
```

   - **Eg:**

```
struct addr
{
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
};
```

   Different types of structure declarations with examples

   1. **Structure declaration only (no variable)**
      In this method, only the structure type or template is defined using a structure tag. No memory is allocated at this stage because no structure variable is created. The declaration acts as a blueprint that specifies the form of the data. Variables of this structure type can be declared later in the program using the structure tag. This method is useful when multiple variables of the same structure type are required at different places in the program.
      Syntax:

```
struct tag {
        type member-name;
        type member-name;
        type member-name;
        type member-name;
        .
        .
        .
};
```

Eg:

```
struct addr
{
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
};
```

Explanation:

- This statement defines only the structure type.
- No memory is allocated at this stage.
- It acts as a template or blueprint.

2. **Structure declaration with variables**

The second way is structure declaration with variables. Here, the structure is defined and one or more structure variables are declared in the same statement. Memory is immediately allocated for each structure variable. Each variable contains its own separate copy of all structure members, so changes made to one structure variable do not affect the others. This method is convenient when the structure variables are needed immediately after defining the structure.

Syntax:

```
struct tag {
        type member-name;
        type member-name;
        type member-name;
        type member-name;
        .
        .
        .
}structure-variables;
```

Eg:

```
struct addr {
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
} addr_info, binfo, cinfo;
```

Explanation:

- Declares the structure type and creates variables at the same time.
- Memory is allocated for each variable.
- Each structure variable has its own separate copy of members.

3. **Anonymous structure (no tag, with variable)**

The third way is anonymous structure declaration with a variable. In this case, the structure has no tag name, and only a single structure variable is declared.

Since the structure has no name, it cannot be reused to create additional variables later. Memory is allocated only for the declared variable. This method is used when only one instance of the structure is required in the entire program.

Syntax:

```
struct {
        type member-name;
        type member-name;
        type member-name;
        type member-name;
        .
        .
        .
}structure-variable;
```

Eg:

```
struct {
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
} addr_info;
```

Explanation:

- Structure has no tag name (anonymous).
- Only one structure variable is created.
- The structure type cannot be reused.

2. **Explain structure with an example. How to access the structure members? How to assign one structure to the another?**
   - A structure is a collection of variables referenced under one name, providing a convenient means of keeping related information together.
   - A structure declaration forms a template that can be used to create structure objects (that is, instances of a structure).
   - The variables that make up the structure are called members. (Structure members are also commonly referred to as elements or fields.)
   - The members of a structure are logically related.
   - The general form of a structure declaration is:

```
struct tag {
   type member-name;
   type member-name;
   type member-name;
   .
   .
   .
} structure-variables;
```

- **Eg:**

```
struct addr
{
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
};
```

- Individual members of a structure are accessed through the use of the . operator (usually called the dot operator).
- The general form for accessing a member of a structure is object-name.member-name;
- In order to access the zip variable: addr_info.zip=123456;
- In order to print the zip variable: printf("%d",addr_info.zip);
- The information contained in one structure can be assigned to another structure of the same type using a single assignment statement. It is not necessary to assign the value of each member separately.
- **Eg code:**

```
#include <stdio.h>
int main(void)
{
    struct
    {
        int a;
        int b;
    } x, y;
    x.a = 10;
    x.b = 20;
    y = x;
    printf("%d\n", y.a);
    return 0;
}
```

This program defines an anonymous structure with two integer members and creates two variables, x and y. The values 10 and 20 are assigned to the members of x using the dot operator. The statement y = x; copies all the member values of x into y. Finally, the program prints the value of y.a, which is 10.

3. Illustrate arrays of structures with syntax and an example.

- Structures are often arrayed. To declare an array of structures, the user must first define a structure and then declare an array variable of that type.
- An array of structures is a collection where each element is a structure of the same type. This is useful for storing and managing multiple records efficiently, such as student lists, employee data, or product inventories.
- An array of structures is needed to store and manage a large number of records of the same type. Declaring separate structure variables for each record becomes difficult and impractical when the number of records is large.
- An array of structures stores all related records under a single name, with each element representing one complete record. It allows easy access using index values, supports the use of loops for processing data, reduces program complexity, and improves code readability.
- **Array of structures declaration:** struct struct_name arr_name [size];
- **Eg:** struct addr_info[3];

- To access the members of a structure stored in an array, the array index is used along with the dot (.) operator. This allows access to a specific structure element and then to one of its members.
- **The general syntax is:** arr_name[index].member_name
  where:
  - arr_name represents the array of structures
  - index specifies the position of a particular structure in the array
  - member_name refers to the specific member of that structure.
- **Eg:** addr_info[1].name;
- **Eg code:**

```c
#include <stdio.h>
struct {
    char name[30];
    char street[40];
    char city[20];
    char state[3];
    unsigned long int zip;
} addr_info[3];
int main() {
    for (int i = 0; i < 3; i++) {
        printf("Enter name, city and zip:\n");
        scanf("%s %s %lu",
              addr_info[i].name,
              addr_info[i].city,
              &addr_info[i].zip);
    }
    for (int i = 0; i < 3; i++) {
        printf("\nName: %s City: %s Zip: %lu",
               addr_info[i].name,
               addr_info[i].city,
               addr_info[i].zip);
    }
    return 0;
}
```

This program declares an array of structures to store address details of three persons. A loop is used to input the name, city, and zip code for each structure in the array. Another loop displays the stored information by accessing structure members using the array index and dot operator.

4.

5. Illustrate the use of the typedef keyword in C with syntax and an example program.

- The typedef keyword in C is used to assign a new, meaningful name to an existing data type. It does not introduce a new data type into the language; instead, it creates an alias for an already defined type. Both the original type name and the new name can be used interchangeably throughout the program.
- One of the major advantages of using typedef is portability. Different computer systems may use different sizes or representations for certain data types. By defining machine-dependent types using typedef, the program becomes easier to adapt to new environments. When the program is moved to another machine, only the typedef declarations need to be updated, while the rest of the code remains unchanged. This reduces effort and minimizes errors during porting.
- **The general syntax of a typedef statement is:**
  typedef type newname;

Here, type represents any valid existing data type, such as int, float, struct, or pointer types, and newname is the alias created for that type. The alias does not replace the original type; it simply provides an additional name that can be used in declarations.

- For example, typedef float balance; declares balance as another name for the float data type. After this definition, variables can be declared using balance, such as: balance over_due;
- In this case, over_due is a floating-point variable, but the name balance makes the purpose of the variable clearer and improves code readability.
- A typedef name can also be used to create another alias. For instance,
  typedef balance overdraft;
- defines overdraft as another name for balance, which ultimately refers to float. Even though multiple names exist, all of them represent the same underlying data type. No new memory layout or behavior is introduced.
- In addition to basic data types, typedef is commonly used with structures, pointers, and function pointers to simplify complex declarations. This leads to clearer, more self-documenting code that is easier to understand and maintain.
- **Eg code:**

```c
#include <stdio.h>
// typedef for a basic data type
typedef float balance;
// typedef using another typedef name
typedef balance overdraft;
int main() {
    balance acc_balance;
    overdraft due_amount;
    acc_balance = 2500.75;
    due_amount = 500.50;
    printf("Account Balance: %.2f\n", acc_balance);
    printf("Overdraft Amount: %.2f\n", due_amount);
    return 0;
}
```

This program uses typedef to create new names (balance and overdraft) for the float data type. Variables are declared using these typedef names and assigned floating-point values. The program prints the values, showing that typedef improves readability without creating a new data type.

6. **Develop a C program to access and modify the members of structures, in an array of structures in C.**

```c
#include <stdio.h>
/* Define a structure */
struct Student {
    int roll;
    char name[20];
    float marks;
};
int main() {
    struct Student s[3];
    int i;
    /* Accessing and storing values */
    for (i = 0; i < 3; i++) {
        printf("\nEnter details of student %d\n", i + 1);

        printf("Roll number: ");
        scanf("%d", &s[i].roll);

        printf("Name: ");
        scanf("%s", s[i].name);

        printf("Marks: ");
        scanf("%f", &s[i].marks);
    }
    /* Modifying a structure member */
    s[1].marks = 95.5;

    /* Displaying the structure data */
    printf("\nStudent Details:\n");
    for (i = 0; i < 3; i++) {
        printf("\nStudent %d\n", i + 1);
        printf("Roll: %d\n", s[i].roll);
        printf("Name: %s\n", s[i].name);
        printf("Marks: %.2f\n", s[i].marks);
    }
    return 0;
}
```

7. **Define a structure with a name student. Develop a C program that uses a structure named student. The program should read and display the details of 'N' students, compute the average marks of the class, and identify the students who have scored marks above and below the class average.**

```c
#include <stdio.h>
struct student {
    int roll;
    float marks;
};
int main() {
    struct student s[10];
    int i, n;
    float sum = 0, avg;
    printf("Enter number of students: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("\nEnter roll number and marks of student %d: ", i + 1);
        scanf("%d %f", &s[i].roll, &s[i].marks);
        sum += s[i].marks;
    }
    avg = sum / n;
    printf("\nClass Average = %.2f\n", avg);
    printf("\nAbove Average Students:\n");
    for (i = 0; i < n; i++) {
        if (s[i].marks > avg)
            printf("Roll: %d  Marks: %.2f\n", s[i].roll, s[i].marks);
    }
    printf("\nBelow Average Students:\n");
    for (i = 0; i < n; i++) {
        if (s[i].marks < avg)
            printf("Roll: %d  Marks: %.2f\n", s[i].roll, s[i].marks);
    }
    return 0;
}
```

8.