Defining Structure:

A structure in C is a user-defined data type that allows you to group together variables of different data types under a single name. The syntax for defining a structure is:

c

Copy code

```
struct structure_name {
// Member variables
data_type member1;
data_type member2;
// ...
};
```

Declaration of Structure Variable:

After defining a structure, you can declare variables of that structure type:

c

Copy code

```
struct structure_name variable_name;
```

Accessing Structure Members:

You can access structure members using the dot (.) operator:

c

Copy code

```
variable_name.member1 = value;
```

Copying and Comparing Structure Variables:

You can copy structure variables using the assignment operator (=):

c

Copy code

```
struct structure_name variable2 = variable1;
```

To compare structure variables, you can use comparison operators (==, !=) or memcmp() function.

Operation on Individual Members:

You can perform operations on individual structure members like any other variable:

c

Copy code

```c
variable_name.member1 += 10;
```

Nesting of Structures:

You can nest structures within other structures:

c

Copy code

```c
struct inner_structure {
int inner_member;
};

struct outer_structure {
struct inner_structure inner;
int outer_member;
};
```

Array of Structures:

You can create arrays of structures:

c

Copy code

```c
struct structure_name array_name[size];
```

Application of Pointers and Functions on Structures:

Pointers can be used to manipulate structures dynamically, allocate memory for structures, and pass structures to functions efficiently.

Union:

A union is similar to a structure, but it allows you to store different data types in the same memory location. The memory size allocated for a union is equal to the size of its largest member.

Static and Dynamic Memory Allocation:

Static Allocation: Memory for variables is allocated at compile time and remains fixed throughout the program's execution.

Dynamic Allocation: Memory is allocated at runtime using functions like malloc(), calloc(), realloc(), and free(). This allows for flexible memory management.

C Program Based on Above Concepts:

A C program based on these concepts might include:

Definition of structures representing various entities (e.g., student, employee).

Declaration and initialization of structure variables.

Accessing, copying, and comparing structure variables.

Operations on individual structure members.

Nesting of structures and arrays of structures.

Application of pointers and functions for efficient manipulation of structures.

Demonstration of static and dynamic memory allocation using structures.