CL1002 ***Programming Fundamentals Lab***	Lab 11 Nested Structures, Composition and Structure array and Filing in C

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2024

Author: Mr. Muhammad Khalid

AIMS AND OBJECTIVES

Aim:

To understand and implement nested structures, structure composition, arrays of structures, and file handling to manage and organize complex data in C programming.

Objectives:

• Understand and implement nested structures to represent hierarchical data types within structures.
• Learn composition to combine multiple structures and achieve modular, organized data modeling.
• Explore the use of arrays of structures to manage collections of related data entries efficiently.
• Implement file handling to read from and write to files, enabling persistent storage and retrieval of structured data in C.

• Practice accessing and manipulating structure members using pointers and array indexing to strengthen data management skills.

Introduction

Structures are derived data types—they're constructed using objects of other types. Normally, we use structure to store the record or the details of any item or entity. Structure members can be variables of primitive data types (e.g., int, float, etc.), or aggregates, such as arrays and other structures.

Keyword **struct** introduces a structure definition

The identifier Chocolate is the structure tag, which names the structure definition and is used with struct to declare variables of the structure type e.g., struct Chocolate kitkat, Mars, Jubilee. Variables declared within the braces of the structure definition are the structure's members. Members of the same structure type must have unique names, but two different structure types may contain members of the same name without conflict.

Section 1: Structures

The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type. The **struct** keyword is used to define the structure in the C programming language. The items in the structure are called its **member** and they can be of any valid data type. Additionally, the values of a structure are stored in contiguous memory locations like arrays. The difference between an array and a structure is that an array is a homogenous collection of similar types, whereas a structure can have elements of different types stored adjacently and identified by a name.

1.1 Declaration of Struct

```
struct Choclate{
char Name[20];
float Weight;
int Calories;
float Price;
char ExpiryDate[10];
};
```

1.2 Declaration & Initialization of Struct type Variables

You can declare the variables before the semi-colon(;) or using a proper declaration syntax like other

variable’s in main();

```
struct Choclate{
char Name[20];
float Weight;
int Calories;
float Price;
char ExpiryDate[10];
}var1,var2,var3;

int main()
{
Struct Choclate Kitkat, Mars, Jubliee, mychoclate[3];
}
```

// OR

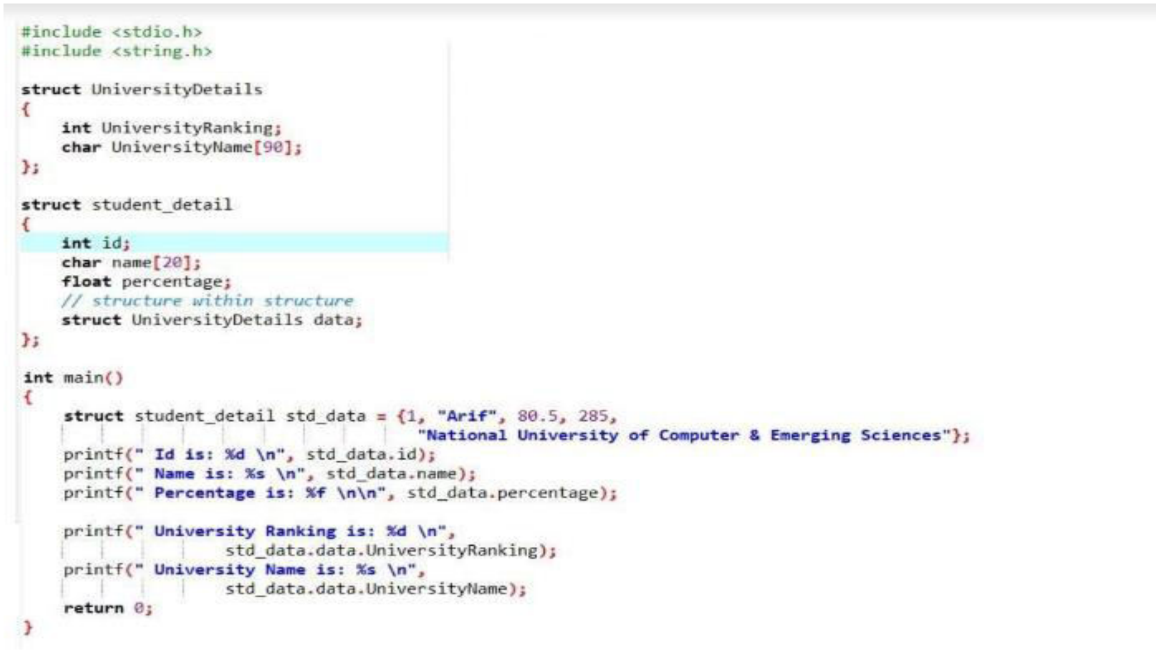
```
struct Choclate myChoclate;
gets(myChoclate.Name);
myChoclate.Weight= 20;
myChoclate.Calories= 500;
myChoclate.Price= 100;
strcpy(myChoclate.ExpiryDate,"01-Feb-2021");
// OR
struct Choclate Jubliee = {"Jubilee",20.50,500,100,"01-Feb-2021"};
}
```

1.3 Declaration & Initialization of Struct type Array

```
int main()
{
// Array of Struct
struct Choclate myFavChoclates[3]; // It is an array of sruct
int i = 0;
while(i<3)
{
gets(myChoclate[i].Name);
scanf("%f",&myChoclate.Weight);
scanf("%d",&myChoclate.Calories);
scanf("%f",&myChoclate.Price);
gets(myChoclate[i].ExpiryDate);
++i;
}
// TO print this array of Struct
i = 0;
while(i<3)
{
puts(myChoclate[i].Name);
printf("%f",myChoclate.Weight);
printf("%d",myChoclate.Calories);
printf("%f",myChoclate.Price);
puts(myChoclate[i].ExpiryDate);
++i;
}
}
```

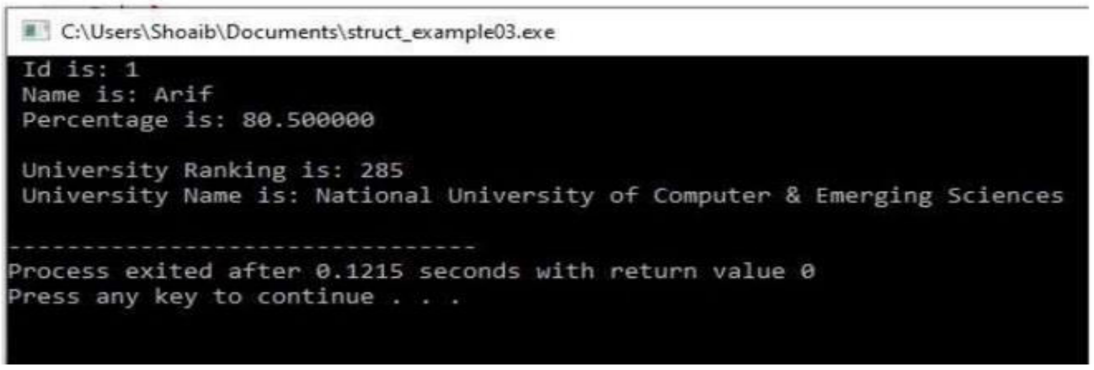
2.0 Nested Structures

Nested structure in C is nothing but structure within structure. One structure can be declared inside
other structure as we declare structure members inside a structure. The structure variables can be a
normal structure variable ,array or a pointer variable to access the data. You can learn below concepts
in this section.



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

struct UniversityDetails
{
    int UniversityRanking;
    char UniversityName[90];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct UniversityDetails data;
};

int main()
{
    struct student_detail std_data = {1, "Arif", 80.5, 285,
                                     "National University of Computer & Emerging Sciences"};
    printf(" Id is: %d \n", std_data.id);
    printf(" Name is: %s \n", std_data.name);
    printf(" Percentage is: %f \n\n", std_data.percentage);

    printf(" University Ranking is: %d \n",
                std_data.data.UniversityRanking);
    printf(" University Name is: %s \n",
                std_data.data.UniversityName);
    return 0;
}
```

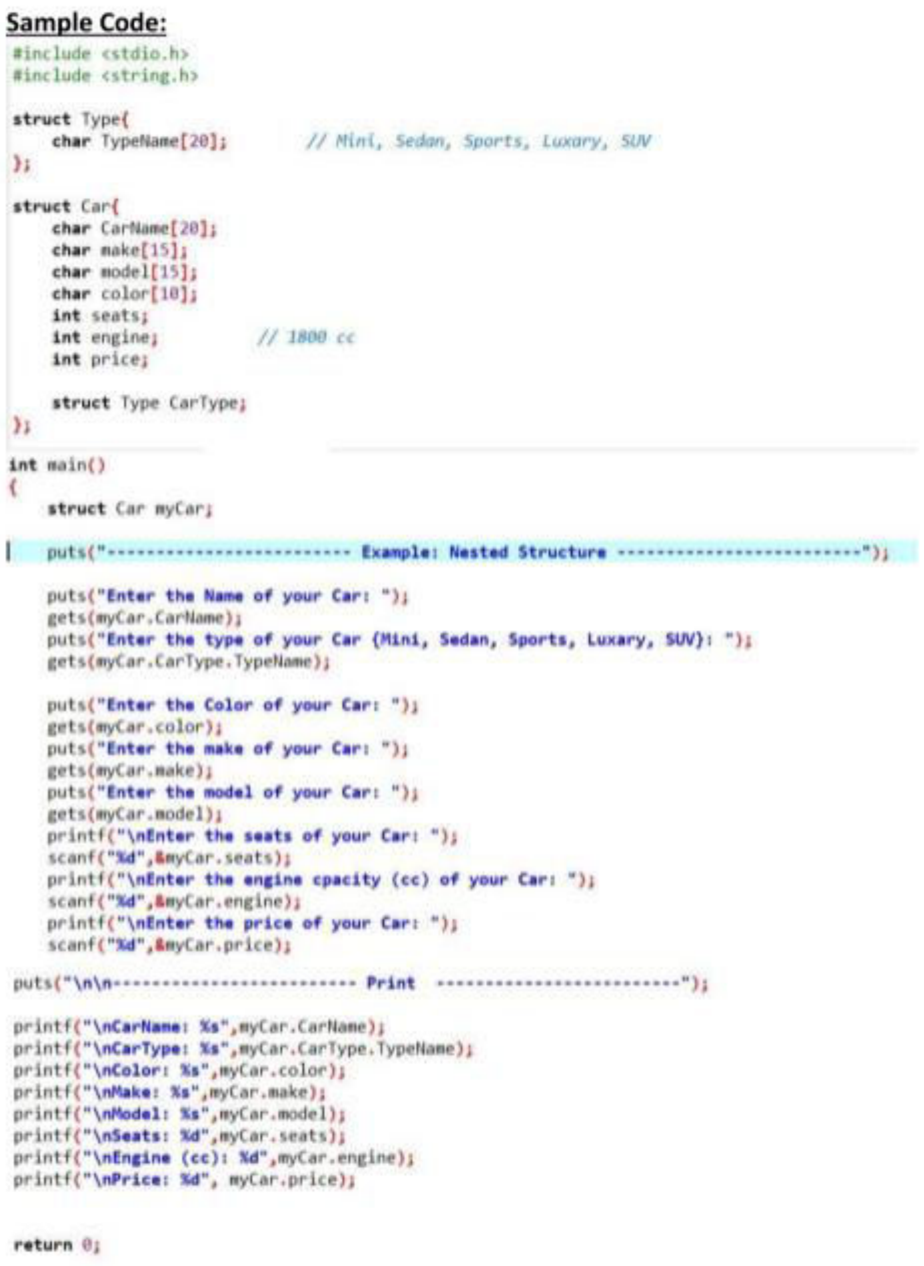
OUTPUT:



C:\Users\Shoaib\Documents\struct_example03.exe

```
 Id is: 1
 Name is: Arif
 Percentage is: 80.500000

 University Ranking is: 285
 University Name is: National University of Computer & Emerging Sciences

--------------------------------
Process exited after 0.1215 seconds with return value 0
Press any key to continue . . .
```

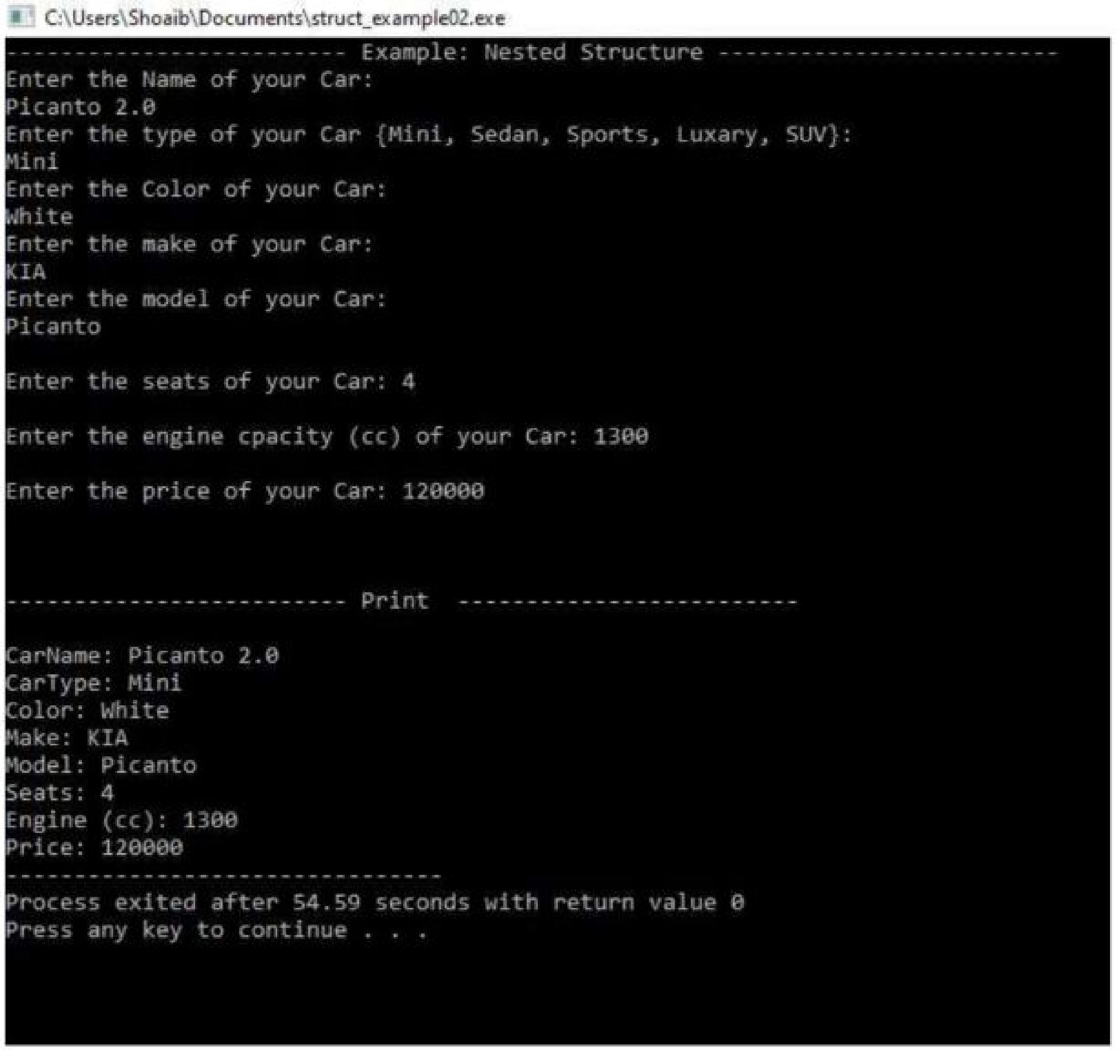
Another Examples of nested structures



Sample Code:

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

struct Type{
    char TypeName[20];        // Mini, Sedan, Sports, Luxary, SUV
};

struct Car{
    char CarName[20];
    char make[15];
    char model[15];
    char color[10];
    int seats;
    int engine;         // 1800 cc
    int price;

    struct Type CarType;
};

int main()
{
    struct Car myCar;

    puts("------------------------ Example: Nested Structure ------------------------");

    puts("Enter the Name of your Car: ");
    gets(myCar.CarName);
    puts("Enter the type of your Car (Mini, Sedan, Sports, Luxary, SUV): ");
    gets(myCar.CarType.TypeName);

    puts("Enter the Color of your Car: ");
    gets(myCar.color);
    puts("Enter the make of your Car: ");
    gets(myCar.make);
    puts("Enter the model of your Car: ");
    gets(myCar.model);
    printf("\nEnter the seats of your Car: ");
    scanf("%d",&myCar.seats);
    printf("\nEnter the engine cpacity (cc) of your Car: ");
    scanf("%d",&myCar.engine);
    printf("\nEnter the price of your Car: ");
    scanf("%d",&myCar.price);
puts("\n\n------------------------ Print  ------------------------");

printf("\nCarName: %s",myCar.CarName);
printf("\nCarType: %s",myCar.CarType.TypeName);
printf("\nColor: %s",myCar.color);
printf("\nMake: %s",myCar.make);
printf("\nModel: %s",myCar.model);
printf("\nSeats: %d",myCar.seats);
printf("\nEngine (cc): %d",myCar.engine);
printf("\nPrice: %d", myCar.price);


return 0;
```

OUTPUT:



C:\Users\Shoaib\Documents\struct_example02.exe

```
------------------------- Example: Nested Structure -------------------------
Enter the Name of your Car:
Picanto 2.0
Enter the type of your Car {Mini, Sedan, Sports, Luxary, SUV}:
Mini
Enter the Color of your Car:
White
Enter the make of your Car:
KIA
Enter the model of your Car:
Picanto

Enter the seats of your Car: 4

Enter the engine cpacity (cc) of your Car: 1300

Enter the price of your Car: 120000



------------------------- Print  -------------------------

CarName: Picanto 2.0
CarType: Mini
Color: White
Make: KIA
Model: Picanto
Seats: 4
Engine (cc): 1300
Price: 120000
--------------------------------
Process exited after 54.59 seconds with return value 0
Press any key to continue . . .
```

C File Handling

File Handling (Filling) refers to working with files for storing data permanently. The standard library provides functions for reading from and writing to files, managing files, and controlling their state. Here’s a basic guide on file handling in C.

1. File Pointers

A file pointer is required to work with files in C. The FILE type (defined in stdio.h) is used to declare a file pointer:

```
FILE *filePointer;
```

2. Opening a File

To open a file, use the fopen() function, which returns a file pointer:

```
FILE *filePointer = fopen("filename.txt", "mode");
```

Common modes include:

- `"r"` - Read (file must exist).
- `"w"` - Write (creates or overwrites the file).
- `"a"` - Append (creates if file does not exist).
- `"r+"` - Read and write (file must exist).
- `"w+"` - Read and write (overwrites file).
- `"a+"` - Read and write (appends to file).

Example:

```
FILE *filePointer = fopen("data.txt", "w");
if (filePointer == NULL) {
    printf("Error opening file\n");
    return 1;
}
```

3. Writing to a File

To write data, use fprintf() or fputc() for characters

```c
fprintf(filePointer, "Hello, World!\n");
fputc('A', filePointer);
```

- Use **`fprintf()`** when you need to write formatted text or a variety of data types in one line.
- Use **`fputc()`** for writing single characters, such as when outputting individual characters in a loop.

Example comparison:

```c
FILE *filePointer = fopen("example.txt", "w");

// Using fprintf to write a formatted string
fprintf(filePointer, "Hello, number: %d\n", 123);

// Using fputc to write each character separately
fputc('H', filePointer);
fputc('i', filePointer);
fputc('\n', filePointer);

fclose(filePointer);
```

4. Reading from a File

To read data, use fscanf() for formatted data or fgetc() for characters:

```c
fscanf(filePointer, "%d", &number);
char ch = fgetc(filePointer);
```

- **Use `fscanf()`** when you want to read formatted or structured data, such as numbers or strings separated by spaces or newlines.
- **Use `fgetc()`** when you need to process data character by character, such as for character-by-character analysis, line counting, or reading unformatted text.

Example Comparing Both

Here's a quick comparison, suppose data.txt contains the text:

```
123 Hello
```

Using fscanf():

```
FILE *filePointer = fopen("data.txt", "r");
int num;
char word[10];
fscanf(filePointer, "%d %s", &num, word);
printf("Number: %d, Word: %s\n", num, word);
fclose(filePointer);


//Output = Number: 123, Word: Hello
```

Using fgetc():

```
FILE *filePointer = fopen("data.txt", "r");
int ch;
while ((ch = fgetc(filePointer)) != EOF) {
    putchar(ch);  // Print each character
}
fclose(filePointer);


//Output = 123 Hello
```

5. Closing a File

Always close the file with `fclose()` to free resources :

```
fclose(filePointer);
```

Note: When you open a file in Read-Only mode “r” you must close the file in order to write into the file or using the “r+”, “w+” or “a+” modes.

6. Binary Mode

When working with files in C, you can open files in **binary mode** or **text mode**. Binary mode is specified by adding a "b" to the mode string (e.g., "rb", "wb", "ab").

Why Binary Mode?

In binary mode:

- The data is read/written in its raw, byte-for-byte format, with no conversions.
- No newline translations happen (unlike text mode, where some systems convert "\n" to "\r\n" when writing to a file).
- All bytes are read as they are, making it ideal for non-text data, like images, executables, and files with specific byte structures.

Text mode, on the other hand, might interpret certain bytes as control characters (e.g., newline characters) and modify them, which is generally undesirable when dealing with binary data.

Example of Binary Mode Usage

Suppose you have a structure you want to save to and load from a binary file:

```c
#include <stdio.h>

struct Person {
    int id;
    char name[20];
};

int main() {
    FILE *file;
    struct Person person = {1, "Alice"};

    // Writing to a binary file
    file = fopen("people.dat", "wb");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    fwrite(&person, sizeof(struct Person), 1, file);
    fclose(file);

    // Reading from the binary file
    file = fopen("people.dat", "rb");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    struct Person readPerson;
    fread(&readPerson, sizeof(struct Person), 1, file);
    printf("ID: %d, Name: %s\n", readPerson.id, readPerson.name);
    fclose(file);

    return 0;
}
```

In this example:

- The file is opened in binary mode ("wb" for writing and "rb" for reading).
- The fwrite() and fread() functions work directly with the bytes in memory.
- fseek() could be used here to navigate through multiple Person records in the file by setting the pointer to specific positions, making it efficient for large files.

Using fseek() in Binary Mode:

When dealing with binary files, fseek() is often combined with ftell() (which returns the current file pointer position) for random access to data. For example, if a file contains a sequence of fixed-size records, fseek() can jump directly to any record by using the record's offset.

Example:

```
int fseek(FILE *stream, long offset, int origin);
```

Parameters:
stream: A file pointer to the file opened with fopen().
offset: Number of bytes to move the pointer from the position specified by origin.
origin: The starting point for the offset. Possible values are:
SEEK_SET: Start of the file.
SEEK_CUR: Current position of the file pointer.
SEEK_END: End of the file.
Return Value : fseek() returns 0 on success and a non-zero value on failure.

Another Example

Suppose each Person record is exactly sizeof(struct Person) bytes. You can jump to a specific record like this:

```
int recordNumber = 5; // Jump to the 5th record (zero-based index)
fseek(file, recordNumber * sizeof(struct Person), SEEK_SET);
```

This allows efficient, direct access to the data without reading the entire file sequentially.

Another Example:

```
FILE *file = fopen("data.bin", "rb");
if (file == NULL) {
    printf("Error opening file.\n");
    return 1;
}

// Move to the 10th byte from the start of the file
fseek(file, 10, SEEK_SET);
int data = fgetc(file);  // Read a byte after moving the pointer
printf("Data at byte 10: %d\n", data);

fclose(file);
```

In this example, fseek() moves the file pointer 10 bytes from the start, allowing direct access to a specific part of the file.

Problems

1. Write a program to copy the contents of one file to another. (Create a File with some dummy data, The data is not required to be formatted)

2. How would you implement a program to count the occurrences of each word in a text file? Write the approach and code.

3. Create a function to replace a word in a file with another word.
   Read the file into memory, perform a search and replace, and write the modified content back to the file.

4. Create a program that defines a structure Student with fields roll_no, name, and another nested structure Marks (fields: maths, science, and english). Use an array of Student to store data for 5 students and calculate the average marks for each student.

5. Define a structure Date with fields day, month, and year. Create another structure Event with fields event_name, date (nested Date structure), and location. Write a program to display all event details in a readable format.

6. Define a structure Employee with fields name, id, salary, and a nested structure Address (fields: city and zip_code). Write a program to input details for multiple employees, store them in a file, and read them back to display.

7. Define a structure Company with fields name, year_established, and an array departments containing the names of up to 5 departments. Write a program to display company details including all departments.