CL1002	Lab 11
Programming Fundamentals Lab	Nested Structures, and Filing in C

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2024

Author: Mr. Muhammad Nouman Hanif

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 the 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;
}
```
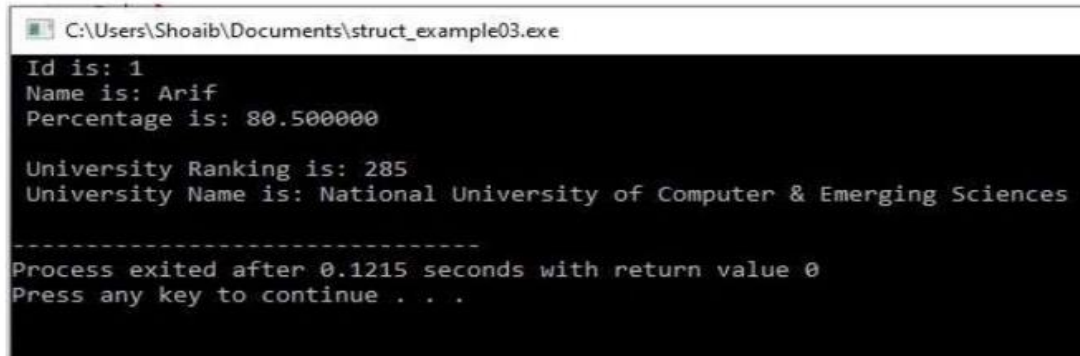
OUTPUT:

```
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:

```c
#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 is a container in computer storage devices used for storing data.

Why files are needed?
- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all.
- However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of Files:
When dealing with files, there are two types of files you should know about:
1. Text files
2. Binary files

1. Text files
Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad. When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents. They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

2. Binary files
Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold a higher amount of data, are not readable easily, and provide better security than text files.

File Operations
In C, you can perform four major operations on files, either text or binary:
1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Opening a file is performed using the **fopen()** function defined in the **stdio.h** header file. The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode");
```

For example,

```
fopen("E:\\cprogram\\newprogram.txt","w");


fopen("E:\\cprogram\\oldprogram.bin","rb");
```

Let's suppose the file **newprogram.txt** doesn't exist in the location **E:\cprogram**. The first function creates a new file named newprogram.txt and opens it for writing as per the mode **'w'.** The writing mode allows you to create and edit (overwrite) the contents of the file.

Now let's suppose the second binary file **oldprogram.bin** exists in the location **E:\cprogram.** The second function opens the existing file for reading in binary mode **'rb'.** The reading mode only allows you to read the file, you cannot write into the file.

Opening Modes in Standard I/O		
Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, `fopen()` returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, `fopen()` returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. Data is added to the end of the file.	If the file does not exist, it will be created.
ab	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	If the file does not exist, `fopen()` returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, `fopen()` returns NULL.

w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Closing a File

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using the **fclose()** function.

```
fclose(fptr);
```

Here, fptr is a file pointer associated with the file to be closed.

Reading and writing to a text file

For reading and writing to a text file, we use the functions **fprintf()** and **fscanf()**. They are just the file versions of **printf()** and **scanf()** . The only difference is that **fprintf()** and **fscanf()** expects a pointer to the structure FILE.

Example 1: Write to a text file.

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

int main()
{
    int num;
    FILE *fptr;

    // use appropriate location if you are using MacOS or Linux
    fptr = fopen("C:\\program.txt","w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);

    return 0;
}
```

This program takes a number from the user and stores it in the file program.txt. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open the file, you can see the integer you entered.

Example 2: Read from a text file.

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

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt","r")) == NULL){
        printf("Error! Opening file");

        // Program exits if the file pointer returns NULL.
        Exit(1);
    }

    fscanf(fptr,"%d", &num);

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}
```

This program reads the integer present in the program.txt file and prints it onto the screen. If you successfully created the file from Example 1, running this program will get you the integer you entered. Other functions like **fgetchar(), fputc()** etc. can be used in a similar way.

Reading and writing to a binary file

Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the fwrite() function. The functions take four arguments:

1. address of data to be written in the disk

2. size of data to be written in the disk

3. number of such type of data

4. pointer to the file where you want to write.

fwrite(addressData, sizeData, numbersData, pointerToFile);

Example 3: Write to a binary file using fwrite()

```c
#include <stdio.h>
#include <stdlib.h>
struct threeNum {
   int n1, n2, n3; };
int main() {
   int n;
   struct threeNum num;
   FILE *fptr;

   // Corrected file path and parentheses
   if ((fptr = fopen("C:\\program.bin", "wb")) == NULL) {
      printf("Error! opening file\n");
      // Program exits if the file pointer returns NULL.
      exit(1); }
   for (n = 1; n < 5; ++n) {
      num.n1 = n;
      num.n2 = 5 * n;
      num.n3 = 5 * n + 1;
      fwrite(&num, sizeof(struct threeNum), 1, fptr);
   }
   fclose(fptr);
   return 0; }
```

Example 4: Read from a binary file using fread()

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

struct threeNum
{
    int n1, n2, n3;
};

int main()
{
    int n;
    struct threeNum num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    for(n = 1; n < 5; ++n)
    {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
    }
    fclose(fptr);

    return 0;
}
```

Problems

Task 01: Design a structure to store information about students, including personal details, academic details, and address. Write a program to input details for 5 students and display the student with the highest GPA. **Hint:** Use a nested structure for personal details and address within the student structure.

Task 02: Create a program to manage employee records in a company. Store employee details in an array of structures and implement the following:
- Add new employees.
- Search for an employee by ID.
- Update salary details for a specific employee.
- Display all employee details sorted by their joining date.

Task 03: Develop a grading system using text files. Write a program to input and store student names, roll numbers, and grades into a text file. Allow the user to view records of students with a grade above a certain threshold.

Task 04: Write a program to store information about books in a library using a binary file. Implement the following operations:
- Add new books.
- Search for a book by title or author.
- Display all books sorted by year of publication.

Task 05: Create a banking system using structures to store account holder details and transaction history. Implement the following features:
- Add new accounts.
- Perform deposits and withdrawals.
- Display the account with the highest balance.
- Save and retrieve the account data from a binary file.

Task 06: Design a nested structure for employees that stores personal details and a timesheet for each day of the week (e.g., hours worked). Write a program to Input timesheet data for 10 employees. Calculate and display the total hours worked by each employee in the week. Identify employees who worked overtime (above 40 hours).

Task 07: Create a system to manage hospital patients using nested structures. Store patient details, including admission and discharge dates. Calculate the duration of each patient's stay in the hospital. Save patient data into a text file and retrieve it.

Task 08: Write a program to manage an inventory of products: Use structures to store product details (ID, name, price, stock quantity). Implement the following:
- Add new products.
- Search for a product by name or ID.
- Update stock quantity after a sale.
- Save and load inventory data from a binary file.

Task 09: Write a program to analyze the performance of a cricket team:
- Use a structure to store player details (name, runs, matches played).
- Read data from a text file.
- Calculate and display the average runs per player.
- Identify the player with the highest runs-to-matches ratio.

Task 10: Create a timetable management system using nested structures:
- Store course details, timings, and instructor details.
- Input and display the timetable for different departments.
- Save the timetable data into a text file.
- Allow the user to retrieve and edit the timetable for a specific department.