Lab 01 Revision of programming concepts of C language

Objectives:

- To revise definition and declaration (instantiation) of Structures in C.
- To revise Pointers in C and their use with Structures.
- To revise file handling in C and writing/reading Structure arrays to/from files.

Reading Task 1: Working with Structures in C

In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.

Defining a Structure:

Before we can create structure variables, we need to define its data type. To define a structure, the ***struct*** keyword is used.

Here is an Example:

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};
```

Here, a derived type ***struct Person*** is defined. Now, we can create variables of this type.

Creating Structure Variables:

When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Here's how we create structure variables:

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};

int main()
{
    struct Person person1, person2, p[20];
    return 0;
}
```

Another way of creating a struct variable is:

```
      struct Person
      {
          char name[50];
          int citNo;
          float salary;
      } person1, person2, p[20];
```

In both cases, two variables person1, person2, and an array variable p having 20 elements of type struct Person are created.

Accessing Members of a structure:

There are two types of operators used for accessing members of a structure.

Dot (.)	- Member operator
Arrow (->)	- Structure pointer operator (will be discussed in the next section)

Suppose, we want to access the salary of person2. Here's how we can do it.

```
person2.salary
```

C Pointers to structures:

Here's how we can create pointers to structures.

```
      struct name
      {
          member1;
          member2;
          .
          .
      };

      int main()
      {
          struct name *ptr;
      }
```

Here, a pointer ***ptr*** of type ***struct name*** is created. That is, ***ptr*** is a pointer to ***struct***.

Access members using Pointer:

To access members of a structure using pointers, we use the Arrow “->” operator.

```c
#include <stdio.h>
struct person
{
   int age;
   float weight;
};
int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;
    printf("Enter age: ");
    scanf("%d", &personPtr->age);
    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);
    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);
    return 0;
}
```

In this example, the address of ***person1*** is stored in the ***personPtr*** pointer using

```c
personPtr = &person1;.
```

Now, we can access the members of ***person1*** using the ***personPtr*** pointer

Dynamic Memory Allocation for structures:

Before you proceed with this section, we recommend you to check C dynamic memory allocation in your programming text book.

Sometimes, the number of struct variables we declared may be insufficient. We may need to allocate memory during run-time. Here's how we can achieve this in C programming.

Example: Dynamic memory allocation of structs

```c
#include <stdio.h>
#include <stdlib.h>
struct person {
   int age;
   float weight;
   char name[30];
};
int main()
{
   struct person *ptr;
   int i, n;
   printf("Enter the number of persons: ");
   scanf("%d", &n);
   // allocating memory for n numbers of struct person
   ptr = (struct person*) malloc(n * sizeof(struct person));
   for(i = 0; i < n; ++i)
   {
       printf("Enter first name and age respectively: ");
       // To access members of 1st struct person,
       // ptr->name and ptr->age is used
       // To access members of 2nd struct person,
       // (ptr+1)->name and (ptr+1)->age is used
       scanf("%s %d", (ptr+i)->name, &(ptr+i)->age);
   }
   printf("Displaying Information:\n");
   for(i = 0; i < n; ++i)
       printf("Name: %s\tAge: %d\n", (ptr+i)->name, (ptr+i)->age);
   return 0;
}
```

When we run the program, the output will be:

```
Enter the number of persons:  2
Enter first name and age respectively:  Harry 24
Enter first name and age respectively:  Gary 32
Displaying Information:
Name: Harry Age: 24
Name: Gary  Age: 32
```

In the above example, ***n*** number of ***struct*** variables are created where ***n*** is entered by the user.

To allocate the memory for n number of struct person, we used,

```
ptr = (struct person*) malloc(n * sizeof(struct person));
```

Then, we used the ***ptr*** pointer to access elements of person.

In-Lab Task 1:

Build and run the program given in Code Listing 1.

In-Lab Task 2:

The program in Code Listing 1 writes a single record to the file. Modify it, to use a function `'write_records_to_file'` with following prototype:

```
int write_records_to_file (struct emp * sptr, int num_records, FILE * fptr)
```

This function should write `'num_records'`' number of structures from a dynamically allocated memory pointed to by `'sptr'` to a file pointed to by `'fptr'`. It should return the number of structures successfully written to the file.

In-Lab Task 3:

Write functions 'read_records_from_file', and 'print_records' with following prototypes

```
int read_records_from_file(struct emp * sptr, int num_records, FILE * fptr)
void print_records(struct emp * sptr, int num_records);
```

Use these functions to read a previously written file (employees_records2.dat) and display the contents on screen.

Post-Lab Task:

The structures that you write in a file using 'fwrite()' are written in the binary format and cannot be viewed in a text editor properly. Your task is to write the contents of these structures in the text format so that the contents may be viewed in a text editor.

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

/// Define a structure 'emp' to hold data about an employee
struct emp
{
   char name[48];  // Name of the employee
   int age;
   float bs;       // Basic Salary as a floating point number.
};

void flush(void);
int write_records_to_file (struct emp * sptr, int num_records, FILE * fptr);
int read_records_from_file(struct emp * sptr, int num_records, FILE * fptr);
void print_records(struct emp * sptr, int num_records);

int main(void)
{
   FILE *fp ;
   char another = 'Y' ;
   struct emp  employee;
   int i = 0;
   int num_rec = 0;

   // Open the file for writing in the Binary Mode
   fp = fopen ( "employees_records.dat", "wb" ) ;

   if ( fp == NULL ){
      printf ( "Cannot open file\n" ) ;
      exit(0) ;
   }

   while ( another == 'Y' )
   {
        printf ( "\nEnter the name of the Employee: " ) ;
        fgets (employee.name, 48, stdin);
        printf ( "\nEnter the age of the Employee: " ) ;
        scanf("%d", &employee.age);
        printf ( "\nEnter the Basic Salary of the Employee: " ) ;
        scanf("%f", &employee.bs ) ;
        // Writing to file
        fwrite ( &employee, sizeof ( struct emp ), 1, fp );

        flush();

        printf ( "Add another record (Y/N) " );

        another = getchar() ;

        flush();

        i++;
    }
  fclose(fp);
  return(0);
}
```

Code Listing 1