Structures in C

By: Atiya Jokhio

Structrues

- Structure is a user-defined datatype in C language which allows us to combine data of different types together.

- struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived datatypes

- Structure helps to construct a complex data type which is more meaningful.

- It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

Structures

- One way of doing this would be creating a different variable for each attribute, however when you need to store the data of multiple students then in that case, you would need to create these several variables again for each student.
- We can solve this problem easily by using structure. We can create a structure that has members for name, id, address and age and then we can create the variables of this structure for each student.

How to create a structure in C Programming

- We use **struct** keyword to create a **structure in C**. The struct keyword is a short form of **structured data type.**

```
struct struct_name {
    DataType member1_name;
    DataType member2_name;
    DataType member3_name;
    …
};
```

- Here struct_name can be anything of your choice. Members data type can be same or different. Once we have declared the structure we can use the struct name as a data type like int, float etc.

How to declare variable of a structure?

```
struct  struct_name  var_name;
```

or

```
struct struct_name {
    DataType member1_name;
    DataType member2_name;
    DataType member3_name;
    …
} var_name;
```

How to access data members of a structure using a struct variable?

```
var_name.member1_name;
var_name.member2_name;
…
```

How to assign values to structure members?

1) Using Dot(.) operator

```
var_name.memeber_name = value;
```

2) All members assigned in one statement

```
struct struct_name var_name =
{value for memeber1, value for memeber2 …so on for all the members}
```

Structures Example

```c
#include <stdio.h>
int main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1, b2, b3 ;
printf ( "\nEnter names, prices & no. of pages of 3 books\n" ) ;
scanf ( "%c %f %d", &b1.name, &b1.price, &b1.pages ) ;
scanf ( "%c %f %d", &b2.name, &b2.price, &b2.pages ) ;
scanf ( "%c %f %d", &b3.name, &b3.price, &b3.pages ) ;
printf ( "\nAnd this is what you entered" ) ;
printf ( "\n%c %f %d", b1.name, b1.price, b1.pages ) ;
printf ( "\n%c %f %d", b2.name, b2.price, b2.pages ) ;
printf ( "\n%c %f %d", b3.name, b3.price, b3.pages ) ;
}
```

Structures Example

```c
#include <stdio.h>
int main( )
{
struct book
{
char name[5] ;
float price ;
int pages ;
} ;
struct book b1={"BOOK", 100.50 ,20};


printf ( "The elements are\n" ) ;
printf ( "\n%s %f %d", b1.name, b1.price, b1.pages ) ;

}
```

Memory allocation of Structure

When the structures are used in the c programming language, the memory does not allocate on defining a structure. The memory is allocated when we create the variable of a particular structure. As long as the variable of a structure is created no memory is allocated. The size of memory allocated is equal to the sum of memory required by individual members of that structure.

```
struct Student{
   char stud_name[30];———— 30 bytes
   int roll_number;———————— 02 bytes
   float percentage;———————— 04 bytes
};                          sum = 36 bytes
```

Partial Initialization

- We can initialize the first few members and leave the remaining blank.
- However, the uninitialized members should be only at the end of the list.
- The uninitialized members are assigned default values as follows:
  - Zero for integer and floating point numbers.
  - '\0' for characters and strings.

```c
#include <stdio.h>
struct student
{
char name[20];
int roll;
char remarks;
float marks;
};
void main()
{
struct student s1={"19k", 11};

printf("Name=%s", s1.name);
printf("\n Roll=%d", s1.roll);
printf("\n Remarks=%c", s1.remarks);
printf("\n Marks=%f", s1.marks);
getch();
}
```

Copying and Comparing Structure Variables

- Two variables of the same structure type can be copied in the same way as ordinary variables.
- If *student1* and *student2* belong to the same structure, then the following statements are valid:

  student1=student2;

  student2=student1;
- However, the statements such as:

  student1==student2

  student1!=student2

  are not permitted.
- If we need to compare the structure variables, we may do so by comparing members individually.

```c
#include <stdio.h>

struct student
{
char name[20];
int roll;
};

void main()
{
struct student student1={"19k", 11 };
struct student student2;

student2=student1;
printf("\nStudent2.name=%s", student2.name);
printf("\nStudent2.roll=%d", student2.roll);
if(strcmp(student1.name,student2.name)==0 && (student1.roll==student2.roll))
{
printf("\n\n student1 and student2 are same.");
}
}
```

Here, structure has been declared global i.e. outside of main() function. Now, any function can access it and create a structure variable.

Array of structure

- Let us consider we have a structure as:

```
struct student
{
char name[20];
 int roll;
char remarks;
float marks;
};
```

- If we want to keep record of 100 students, we have to make 100 structure variables like st1, st2, …,st100.
- In this situation we can use array of structure to store the records of 100 students which is easier and efficient to handle (because loops can be used).

Array of structure

```c
#include <stdio.h>
int main( )
{
struct book
{
char name[10] ;
float price ;
int pages ;
} ;
  struct book b[2];
 int i;
 for (i=0; i<2; i++)
 {
    printf("\nEnter name, price and pages");
    scanf("%s%f%d", &b[i].name, &b[i].price, &b[i].pages);
 }
for (i=0; i<2; i++)
 {
    printf("\n %s %f %d", b[i].name, b[i].price, b[i].pages);
 }}
```

Array of structure

C program to store the roll number , name ,and marks of a student using structure and sort the names in order of maximum marks to minimum marks.

```c
#include<stdio.h>
 struct student
{
    int roll_no,marks;
    char name[25];
}stud[100],t;

void main()
{
    int i,j,n;
    printf("Enter the no of students\n");
    scanf("%d",&n);
    printf("enter student info as roll_no , name , marks\n");
    for(i=0;i<n;i++)
    {
        scanf("%d %s %d",&stud[i].roll_no,stud[i].name,&stud[i].marks);
    }
```

```
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-1;j++)
        {
            if(stud[j].marks<stud[j+1].marks)
            {
                t=stud[j];
                stud[j]=stud[j+1];
                stud[j+1]=t;
            }
        }
    }

    printf("\nStudent info in terms of marks from highest to lowest\n");
    printf("\nROLL_NO\t\tNAME\t\tMARKS\n");
    printf("-------------------------------------------------------------------------\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t\t\t%s\t\t\t%d\n",stud[i].roll_no,stud[i].name,stud[i].marks);
    }
}
```

Question

- Define a structure of employee having data members name, address, age and salary. Take the data for n employees in an array and find the average salary.
- Write a program to read the *name*, *address*, and *salary* of 5 employees using array of structure. Display information of each employee in alphabetical order of their name.

Structure within another Structure (Nested Structure)

- Let us consider a structure personal_recordto store the information of a person as:

- struct personal_record

```
{
char name[20];
int day_of_birth;
int month_of_birth;
int year_of_birth;
float salary;
} person;
```

Structure within another Structure (Nested Structure)…

- In the structure above, we can group all the items related to birthday together and declare them under a substructure as:

```
struct Date
        {
        int day_of_birth;
        int month_of_birth;
        int year_of_birth;
        };

struct personal_record
 {
 char name[20];
 struct Date birthday;
 float salary;
 }person;
```

Structure within another Structure (Nested Structure)…

- Here, the structure personal_record contains a member named birthday which itself is a structure with 3 members. This is called structure within structure.

- The members contained within the inner structure can be accessed as:

  person.birthday.day_of_birth

  person.birthday.month_of_birth

  person.birthday. year_of_birth

- The other members within the structure personal_record are accessed as usual:

  person.name

  person.salary

```
printf("Enter name:\t");
scanf("%s", person.name);
printf("\nEnter day of birthday:\t");
scanf("%d", &person.birthday.day_of_birth);
printf("\nEnter month of birthday:\t");
scanf("%d", &person.birthday.month_of_birth);
printf("\nEnter year of birthday:\t");
scanf("%d", &person.birthday.year_of_birth);
printf("\nEnter salary:\t");
scanf("%f", &person.salary);
```

Structure within another Structure (Nested Structure)…

- ***Note:- More than one type of structures can be nested…***

```
struct  date
{
int day;
int month;
int year;
};

struct name
    {
    char first_name[10];
    char middle_name[10];
    char last_name[10];
    };

struct personal_record
    {
    float salary;
    struct date birthday,deathday;
    struct name full_name;
    };
```

Function with Structure

- **Passing whole structure to functions**

Whole structure can be passed to a function by the syntax:

function_name(structure_variable_name);

The called function has the form:

return_type function_name (struct tag_name structure_variable_name)

{

}

Function with Structure

- **Passing whole structure to functions**

```c
#include<stdio.h>
        struct Employee
        {
                int Id;
                char Name[25];
                int Age;
                long Salary;
        };

        void Display(struct Employee);
        void main()
        {
                struct Employee Emp = {11,"M.Ahmed",25,45000};

                Display(Emp);

        }

  void Display(struct Employee E)
  {
                printf("\n\nEmployee Id : %d",E.Id);
                printf("\nEmployee Name : %s",E.Name);
                printf("\nEmployee Age : %d",E.Age);
                printf("\nEmployee Salary : %ld",E.Salary);
  }
```

Function with Structure

- **Passing array of structures to function**
  - Passing an array of structure type to a function is similar to passing an array of any type to a function.
  - That is, the name of the array of structure is passed by the calling function which is the base address of the array of structure.
  - The function prototype comes after the structure definition.

Function with Structure

- **Passing array of structures to function**

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

struct employee
{
    char name[20];
    int id;
    int age;
    float sal;
};
void Input( struct employee emp[], int count)
{
    int i;
    for(i=0; i<count; i++)
{
    printf("Enter name, Id, age and Salary of an employee");
    scanf("%s %d %d %f", emp[i].name, &emp[i].id, &emp[i].age, &emp[i].sal);
}
Display (emp, count);
}
```

Function with Structure

- **Passing array of structures to function**

```
void Display (struct employee emp[], int count)
{
    int i;
    for (i=0; i<count; i++)
    {
        printf("\n%s is %d years old and has %d id and %.2f Salary.\n",emp[i].name,
        emp[i].age, emp[i].id, emp[i].sal);
    }
}

int main()
{
    struct employee emp[3];
    Input (emp,3);
    return 0;
}
```

Tasks

- Define a structure to store the following student data:
  CGPA, courses (course name, GPA), address (consisting of street address, city, state, zip). Input 2 student records, compare and display which student have highest GPA in which course also Display which student have highest CGPA .
  HINT: define another structure to hold the courses and address.

- Create a struct Rectangle with attributes length and width. Provide functions that calculate the perimeter and the area of the rectangle. The check() function should verify that