Introduction to Structure

http://www.studytonight.com/c/structures-in-c.php

Structure is a user-defined data type in C which allows you to combine different data types to store a particular type of record. Structure helps to construct a complex data type in more meaningful way. It is somewhat similar to an Array. The only difference is that array is used to store collection of similar datatypes while structure can store collection of any type of data.

Structure is used to represent a record. Suppose you want to store record of **Student** which consists of student name, address, roll number and age. You can define a structure to hold this information.

Defining a structure

struct keyword is used to define a structure. **struct** define a new data type which is a collection of different type of data.

Syntax :

```
struct structure_name
{
 //Statements
};
```

Example of Structure

```
struct Book
{
    char name[15];
    int price;
    int pages;
};
```

Here the **struct Book** declares a structure to hold the details of book which consists of three data fields, namely *name*, *price* and *pages*.

These fields are called **structure elements or members**. Each member can have different data type,like in this case, **name** is of `char` type and **price** is of `int` type etc.

Book is the name of the structure and is called structure tag.

Declaring Structure Variables

It is possible to declare variables of a **structure**, after the structure is defined. **Structure** variable declaration is similar to the declaration of variables of any other data types.

Structure variables can be declared in following two ways.

1) Declaring Structure variables separately

```
struct Student
{
    char[20] name;
    int age;
    int rollno;
};

struct Student S1 , S2;    //declaring variables of Student
```

2) Declaring Structure Variables with Structure definition

```
struct Student
{
    char[20] name;
    int age;
    int rollno;
} S1, S2;
```

Here **S1** and **S2** are variables of structure **Student**. However this approach is not much recommended.

Accessing Structure Members

Structure members can be accessed and assigned values in number of ways. Structure member has no meaning independently. In order to assign a value to a structure member, the member name must be linked with the **structure** variable using dot `.` operator also called **period** or **member access** operator.

```
struct Book
{
    char name[15];
    int price;
    int pages;
} b1, b2;
```

```
b1.price=200; //b1 is variable of Book type and price is member of Book
```

We can also use `scanf()` to give values to structure members through terminal.

```
scanf(" %s ", b1.name);
scanf(" %d ", &b1.price);
```

Structure Initialization

Like any other data type, structure variable can also be initialized at compile time.

```
struct Patient
{
    float height;
    int weight;
    int age;
};

struct Patient p1 = { 180.75 , 73, 23 };    //initialization
or,
struct patient p1;
p1.height = 180.75;
p1.weight = 73;          //initialization of each member separately
p1.age = 23;
```

Array of Structure

We can also declare an array of **structure**. Each element of the array representing a **structure** variable. **Example :** `struct employee emp[5];`

The above code define an array **emp** of size 5 elements. Each element of array **emp** is of type **employee**

```
#include< stdio.h>
#include< conio.h>
struct employee
{
 char ename[10];
 int sal;
};

struct employee emp[5];
int i,j;
void ask()
{
 for(i=0;i
```

Nested Structures

Nesting of structures, is also permitted in C language.

Example :

```
struct student
{
    char[30] name;
    int age;
        struct address
        {
            char[50] locality;
            char[50] city;
            int pincode;
        };
};
```

Structure as function arguments

We can pass a structure as a function argument in similar way as we pass any variable or array.

```
/*************************************************************
 * Structure passed as a function argument. The program accepts   *
 * student name and enrollment year and prints them on the screen,*
 * one student at a time.                                         *
 *************************************************************/
#include< stdio.h>
#include< conio.h>
struct student
{
    char name[10];
    int roll;
};
void show(struct student st);
void main()
{
    struct student std;

    printf("\nEnter student record\n");
    printf("\nstudent name\t");
    scanf("%s",std.name);
    printf("\nEnter student roll\t");
    scanf("%d",&std.roll);
    show(std);
    getch();
}

void show(struct student st)
{
    printf("\nstudent name is %s",st.name);
    printf("\nroll is %d",st.roll);
}
```

Often, entering more than one student record is required. Therefore, arrays of structures are needed to be used. The following program is a good example to this, where s[SIZE] is declared as an array of structures. The following title best describes the program which is listed in the next page.

```
/*****************************************************
 * In this program, a structure(student) is created which contain       *
 * name, roll and marks as its data member. Then, an array of structure *
 * of 10 elements is created. Then, data(name, roll and marks) for 4    *
 * elements is asked to user and stored in array of structure. Finally, *
 * the data entered by user is displayed.                               *
 *****************************************************/
```

```c
#include <stdio.h>
#define SIZE 4
struct student{
    char name[50];
    int roll;
    float marks;
};
int main(){
    struct student s[SIZE];
    int i;
    printf("Enter information of students:\n");
    for(i=0;i<SIZE;++i)
    {
        s[i].roll=i+1;
        printf("\nFor roll number %d\n",s[i].roll);
        printf("Enter name: ");
        scanf("%s",s[i].name);
        printf("Enter marks: ");
        scanf("%f",&s[i].marks);
        printf("\n");
    }
    printf("Displaying information of students:\n\n");
    for(i=0;i<SIZE;++i)
    {
     printf("\nInformation for roll number %d:\n",i+1);
     printf("Name: ");
     puts(s[i].name);
     printf("Marks: %.1f",s[i].marks);
   }
   return 0;
}
```

```
/*********************************************************
 * Structure to create a record of person with the persons name *
 * and age entered from the keyboard. The name and nationalities*
 * are defined as pointers where age is an integer.             *
 *********************************************************/

...

struct person
{
    int age;
    char *name;
    char *nation;
};



...


   struct person id;
   char PersonName[15], PersonNation[15];
   int age;
```

```c
#include <stdio.h>
struct person
{
    int age;
    char *name;
    char *nation;
};
main()
{
   struct person id;
   char PersonName[15], PersonNation[15];
   int age;

   printf("Please enter name of the person : ");
   scanf("%s", PersonName);
      printf("Please enter Nationality of the person : ");
      scanf("%s", PersonNation);
        printf("Please enter age of the person : ");
        scanf("%d", &age);
            printf("\n\n");

    id.age = age;
    id.name = PersonName;
    id.nation = PersonNation;

    printf("The name of the person is %s\n", id.name);
    printf("The nationality of the person is %s\n", id.nation);
    printf("The age of the person is %d\n", id.age);
    return 0;
}
```

```
/* The following assignment of a struct to another struct does what one
 * might expect. It is not necessary to use memcpy() to make a duplicate
 * of a struct type. The memory is already given and zeroed by just
 * declaring a variable of that type regardless of member initialization.
 * This should not be confused with the requirement of memory management
 * when dealing with a pointer to a struct.
 */
```

```c
#include <stdio.h>

/* Define a type point to be a struct with integer members x, y */
typedef struct {
   int    x;
   int    y;
} point;
int main(void) {

/* Define a variable p of type point and initialize all its members inline! */
    point p = {1,3};

/* Define a variable q of type point. Members are uninitialized. */
    point q;

/* Assign the value of p to q, copies the member values from p into q. */
    q = p;

/* Change the member x of q to have the value of 3 */
    q.x = 3;

/* Demonstrate we have a copy and that they are now different. */
    if (p.x != q.x) printf("The members are not equal! %d != %d", p.x, q.x);

    return 0;
}
```

```
/* struct_cars.c : An auto gallery keeps the information of the second hand
 * cars by using the following record structure:
   - car brand (e.g. Honda, Suzuki etc.)
   - production year,
   - car type,  (Only two types: 'S': Saloon, 'H': Hatchback)
   - car price.
   Capacity of the gallery is no more than 100 cars. Write a program that,
   (a) defines a new data type called, CARS, to represent the records of
       the cars in the gallery,
   (b) uses a function, ReadCars, to input the records of the cars from the
       keyboard,
   (c) and finally uses another function, SelectedCars, to display the
       cheapest saloon car and the youngest hatchback car in gallery. */

#include <stdio.h>
#define SIZE 3

typedef struct
{
   char brand[20];
    int year;
    char type;
    float price;
} CARS;
```

```
void ReadCars(CARS []);
void SelectedCars(CARS []);

int main()
{
    CARS A[SIZE];

    ReadCars(A);

    SelectedCars(A);

    return 0;
}

void ReadCars(CARS A[])
{
    int i;

    for(i=0; i<SIZE; i++)
    {
        printf("\nEnter Data for the Car Record:");
        printf("\n\n");

        printf("\nEnter Car Brand :");
        scanf("%s",A[i].brand);
```

```
        printf("\nEnter Car Year :");
        scanf("%d",A[i].year);

        printf("\nEnter Car Type (Saloon or Hatchback) :");
        scanf("%c",&A[i].type);

        printf("\nEnter Car Price :");
        scanf("%f",&A[i].price);
    }
}

void SelectedCars(CARS A[])
{
    int i, youngest=3000, ypos, cpos=0;
    float cheapest= 1000000.0;

    for(i=0;i<SIZE;i++)
    {
        if((A[i].type == 'S') && (A[i].price < cheapest))
        {
            cheapest= A[i].price;
            cpos= i;
        }

        if((A[i].type == 'H') && (A[i].year < youngest))
        {
```

```
            // youngest = A[i].price;
            youngest = A[i].year;
            ypos = i;
        }
    }

    printf("Cheapest Saloon Car: %s\n %d\n %c\n %.2f", A[cpos].brand,
A[cpos].year, A[cpos].type, A[cpos].price);
    printf("Youngest Hatch Back Car: %s\n %d\n %c\n %.2f", A[ypos].brand,
A[ypos].year, A[ypos].type, A[ypos].price);
}
```