CSE101-Lec#32-33-34

Structures in functions

Created By:
Amanpreet Kaur &
Sanjeev Kumar
SME (CSE) LPU

Outline

- Passing structure to a function
- Pointers to the structure.

Structure and Function

- The relationship of structure with the function can be viewed from three angles:-
    1. Passing Structures to a function.
    2. Function Returning Structure.
    3. Passing array of Structures to Function.
- Passing multiple arguments in and out of functions through a single argument

Passing Structures to Functions

- Passing structures to functions
  - Pass entire structure
  - Or, pass individual members
    - Both are pass by value.
- To pass structures call-by-reference
  - Pass its address
- To pass arrays by value
  - Create a structure with the array as a member
  - And pass that structure.

Passing Structure to a Function

Similar to passing array of variable, structure can be passed to a function as argument

Syntax

```
type-specifier  func-name(struct-variable);
```

Passing of structure to a function by **value**

```
#include <stdio.h>
struct car{
  char name[50];
  int seats;
  float price;
};
void cardata(struct car); /*function
prototype*/
void main()
{
 struct car myCar = {"Racer", 1, 1200000};
 cardata(myCar); //function calling
}
void cardata(struct car newCar)
{
 printf("\nData about your car is: %s %d %f",
newCar.name, newCar.seats, newCar.price);
}
```

```
Data about your car is Racer 1 1200000
```

```
#include <stdio.h>
struct car{
  char name[50];
  int seats;
  float price;
};
void cardata(struct car); /*function
prototype*/
void main()
{
 struct car myCar = {“Racer”, 1, 1200000};
 cardata(myCar.seats); //function calling
 printf(“\nData about your car is: %s %d %f”,
myCar.name, myCar.seats, myCar.price);
}
void  cardata(struct car newCar)
{
newCar.seats = 2; /*changing the number of
seats*/
}
```

Passing of structure member to a function by **value**.

```
Data about your car is Racer 1 1200000
```

Passing of structure by **reference** to a function

```
#include <stdio.h>
struct car{
   char name[50];
   int seats;
   float price;
};
void cardata(struct car*); /*function
prototype*/
void main()
{
 struct car myCar;
 printf("Enter data:\n");
 cardata(&myCar);
 printf("\nData about your car is: %s %d %f",
myCar.name, myCar.seats, myCar.price);
}
void cardata(struct car *newCar)
{
gets(newCar->name);
scanf("%d %f", &newCar->seats, &newCar->price);
}
```

```
Enter data: Racer 1 1200000
Data about your car is Racer 1 1200000
```

Passing of structure by **reference** to a function

```
#include <stdio.h>
struct car{
   char name[50];
   int seats;
   float price;
};
void cardata(struct car*); /*function
prototype*/
void main()
{
 struct car myCar= {"Racer", 1, 1200000};
 printf("\nData about your car is: %s %d %f",
myCar.name, myCar.seats, myCar.price);

 cardata(&myCar);

 printf("\nData about your car is: %s %d %f",
myCar.name, myCar.seats, myCar.price);
}
void cardata(struct car *newCar)
{
 struct car c = {"Safari", 4, 899000};
 *newCar= c; /*the value of c is copied at
location pointed by newCar*/
}
```

```
Data about your car is Racer 1 1200000
Data about your car is Safari 4 899000
```

Functions returning structures

- It is possible to return a structure from a function
- The **advantage** of returning a structure from a function is where the calling function needs the changes done to the structure without modifying the original contents.

Program to **return** a structure.

```
#include <stdio.h>
struct car{
   char name[50];
   int seats;
   float price;
};
struct car cardata(void); /*function
prototype*/
void main()
{
struct car myCar;
printf("\nEnter data : Name seats and price ");
mycar=cardata();
printf("\nData about your car is %s %d %f\n",
myCar.name, myCar.seats, myCar.price);
}
struct car cardata(void)
{
 struct car newCar;
 scanf("%s %d %f", &newCar.name, &newCar.seats,
&newCar.price);
return newCar;
}
```

```
Enter data: Racer 1 1200000
Data about your car is Racer 1 1200000
```

Array of Structures

- to store data of 100 cars we would be required to use 100 different structure variables from **car1** to **car100**, which is definitely not very convenient. A better approach would be to use an array of structures.

```
    struct car mycar[100];
```

- This provides space in memory for 100 structures of the type **struct car**.

Program to print array of structures.

```
#include <stdio.h>
struct car{
   char name[50];
   int seats;
   float price;
};
void main()
{
 int i;
 struct car myCar[100];

for(i=0; i<100; i++){
   printf("\n\nEnter data for car[%d]:\n", i);
   scanf("%s %d %f", &myCar[i].name,
&myCar[i].seats, &myCar[i].price);
 }

 for(i=0; i<100; i++){
 printf("\nData about your car[%d] is: %s %d
%f", i, myCar[i].name, myCar[i].seats,
myCar[i].price);
 }
}
```

```
Enter data for car0: Racer 1 1200000
Data about your car0 is Racer 1 1200000

Enter data for car1: Micra 4 500000
Data about your car1 is Micra 4 500000

Enter data for car2: RacerGt 1 800000
Data about your car2 is RacerGt 1 800000
.
.
.
.
.
Enter data for car99: RacerEf 1 2000000
Data about your car99 is RacerEf 1 2000000
```

Pointers to Structure

```
struct  car myCar, *ptr;
```

It declares a structures variable *myCar and a* pointer variable *ptr to* structure of type *car.*

ptr can be initialized with the following assignment statement

ptr = &myCar;

HOW WE CAN ACCESS THE ELEMENTS OF STRUCTURE?

```
*ptr.name,*ptr.seats,*ptr.age
```

*But this approach **will not work** because dot has higher priority*

Correctly way to write is:

```
(*ptr).name,(*ptr).seats, (*ptr).price)
```

or

```
ptr->name, ptr->seats, ptr->price
```

Accessing Members of Structures

- Arrow operator (->) used with pointers to structure variables

```
car *myCarPtr = &myCar; //intializing pointer
printf("%s", myCarPtr->name);
```

  - `myCarPtr->name` is equivalent to

    `(*myCarPtr).name`

Program for pointer to a structure

```
#include <stdio.h>
struct car{
       char *name;
       int seats;
       float price;
};
int main()
{
struct car myCar = {“Renault”,2, 500000};
struct car *myCarPtr; //define a pointer to car
myCarPtr = &myCar;  /*assign address of myCar
to myCarPtr */

printf("%s %f %d \n%s %f %d \n%s %f %d\n",
myCar.name, myCar.price, myCar.seats,
myCarPtr->name, myCarPtr->price,
myCarPtr->seats,
(*myCarPtr).name, (*myCarPtr).price,
(*myCarPtr).seats);
} //end main
```

```
Renault 500000 2
Renault 500000 2
Renault 500000 2
```

Next Class: Self Referential Structure

cse101@lpu.co.in