C Programming

Lesson 6: Structures

Table of Contents

- **Structures:**
  - Basics of Structures
    - Nested Structures
    - Structures and Arrays
    - Structures and Pointers
    - Structures and Functions
    - Unions
    - Difference between union and structure
    - Typedef statement
    - Enumerated datatype

Lesson Objectives

- **In this lesson, you will learn:**
  - The concept of structure declaration and definition
  - The method to process structure variables using pointers
  - The method to access the structure members

Lesson Objectives

- **In this lesson, you will cover the following topics:**
  - Structures:
    - Basics of Structures, Declaration
    - Structure Variables, tag, and initialization
    - Accessing Structure Members
  - Nested Structures
  - Arrays of Structures
  - Pointers to Structures, Accessing members with structure pointers
  - Allocating memory for a pointer to a structure
  - Structures as Function Arguments and Function Values

Concept of Arrays of Structures

- **Array contains individual structures as its elements**
- **They are commonly used when a large number of similar records are required to be processed together**
- **For example:**
  - The data of motor containing 1000 parts can be organized in an array of structure as follows:

```
struct item motor[1000];
```

  - It declares motor to be an array containing 1000 elements of the type struct item.

Arrays of Structures – Sample Code

- **Let us see a sample code on "arrays of structures**

```
/* Example- An array of structures */
# include<stdio.h>
void main(void)
{
  struct book
  {
        char name[15];
        int pages;
        float price;
  };
  struct book novel[10];
  int index;
  printf("\n Enter name, pages and price of the book\n");
   /* accessing elements of array of structures */
```

Arrays of Structures – Sample Code (contd.)

```
for(index=0;index<9;index++)
 {
      scanf("%s%d",novel[index].name,&novel[index].pages);
      scanf("%f",&novel[index].price);
      printf("\n");
 }
 printf("\n Name, Pages and Price of the book :\n");
 for(index=0;index<=9;index++)
 {
      printf("%s %d",novel[index].name,novel[index].pages);
      printf("%f",novel[index].price);
 }
}
```

Concept of Arrays within Structures

- **A structure may contain arrays as members**
  - They are used when a string needs to be included in a structure.
  - For example: The structure date can be expanded to include the names of the day of the week and month as shown below:

```
struct date
 {
      char weekday[10];
      int day;
      int month;
      char monthname[10];
      int year;
 };
```

Concept of Arrays within Structures

- **A structure variable today can be declared and initialized as shown below:**

```
struct date today={”Sunday”,1,10,”November”,2011};
```

- **An element of an array contained in a structure can be accessed using the dot and array subscript operators**

```
printf(“%c”,today.monthname[2]);
```

Concept of Pointer to Structures

- **The beginning address of a structure can be accessed by the address of (&) operator.**
- **It is mainly used to create complex data structures such as Linked lists, trees, graphs, and so on**
- **Pointer variables holding address of structure are called Structure Pointers**
  - For example: Consider the following declaration:

```
struct date today,*ptrndate;
```

  - It declares today to be a variable of type struct date, and ptrndate to be a pointer to a struct date variable

Accessing Pointer members in Structures

- **Consider the following structure declaration:**

```
struct account
{
        int acct_no;
        char acct_type;
        char name[20];
        float balance;
        struct date lastpayment;
};
struct account customer,*pc;
```

  - In this example, customer is a structure variable of type account, and pc is a pointer variable whose object is a structure of type account.

Accessing Pointer members in Structures

- **The pointer variable pc can now be used to access the member variables of customer using the dot operator as:**
  - (*pc).acct_no;
  - (*pc).acct_type;
  - (*pc).name;

- **The parentheses are necessary because the dot operator (.) has higher precedence than that of the dereferencing operator(*).**

Accessing Pointer members in Structures

- **The members can also be accessed by using a special operator called structure pointer or arrow operator (->).**
- **The general form for the use of the operator -> is as shown below:**
  - pointer_name->member_name;
- **Thus,**

```
if  pc=&customer;
pc->balance=(*pc).balance=customer.balance
```

  - where, balance is member of structure customer

Structures as Function Arguments

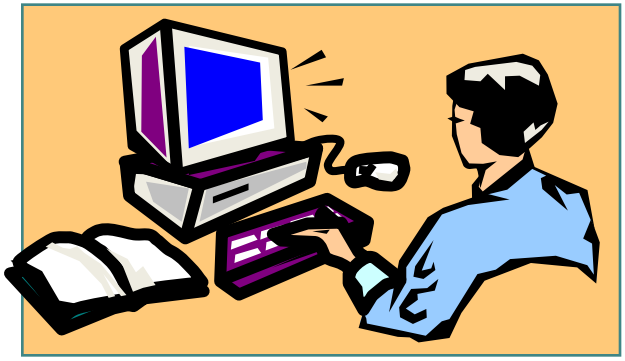
- **C provides three methods of passing structures to a function:**
- **Passing Structure Member to Function**
- **Passing Entire Structure to Function**
- **Passing Structure Pointers to Functions**

#1: Passing Structure Member to Function

- **This method involves supplying structure members as the arguments in a function call.**
- **These arguments are then treated as separate non-structure values, unless they themselves are structures.**
- **Disadvantage:**
  - The relationship between the member variables encapsulated in a structure is lost in the called function.
  - It should only be used if a few structure members need to be passed to the called function.

Demo - Passing Structure Member to function

- **Demo struct_function.c program**



Sample Code

```
/* Example- structure member as function arguments */
# include <stdio.h>
# define CURRENT_YEAR 2011
typedef struct
{
        char name[20];
        struct date
        {
                int day,month,year;
        }birthday;
        float salary;
} emprec;
```

Sample Code (Contd..)

```
/* give increments to employees if age is greater than 30*/
float increment(float sal,int year,int inc)
{
        if(CURRENT_YEAR - year > 30)
                        sal += inc;
        return(sal);
}
void main(void)
{
  int n=500;
  emprec per={"Rohit Tamhane",5,9,1979,4000.50};
  printf(" *** Employee Details ***\n");
  printf("Name :%s \n",per.name);
```

Sample Code (Contd..)

```
printf("Birthdate:%d:%d:%d\n",per.birthday.day,
                per.birthday.month, per.birthday.year);
printf("Salary :%6.2f \n\n",per.salary);
per.salary=increment(per.salary,per.birthday.year,n);
printf(" *** Employee Details *** \n");
printf("Name :%s \n",per.name);
printf("Birthdate: %d:%d:%3d \n",per.birthday.day,
                per.birthday.month, per.birthday.year);
printf("Salary :%6.2f \n",per.salary);
}
```

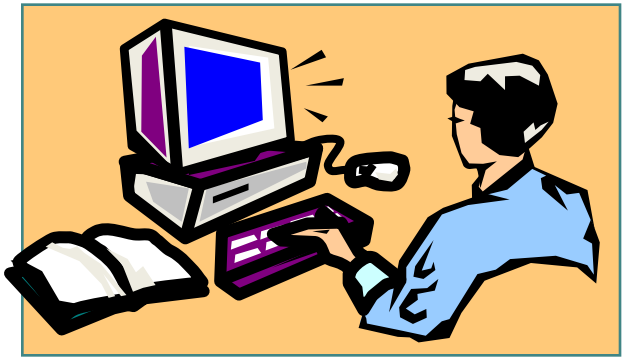
#2: Passing Entire Structure to Function

- **This method involves passing the complete structure to a function by simply providing the name of the structure variable as the argument in the function call.**

- **The corresponding parameter in the called function must be of the same structure type.**

Demo on passing Entire Structure to Function

Demo on Struct_variable_asparameter.c program



Sample Code

```
/* Example- Entire structure as function arguments */
# include<stdio.h>
struct book
{
   char name[20];
   char author[10];
   int pages;
};
void main(void)
{
  void display(struct book);
  struct book tech_book={"Programming in C“,"Stephen“, 300};
  display(tech_book);
}
```

Sample Code contd..

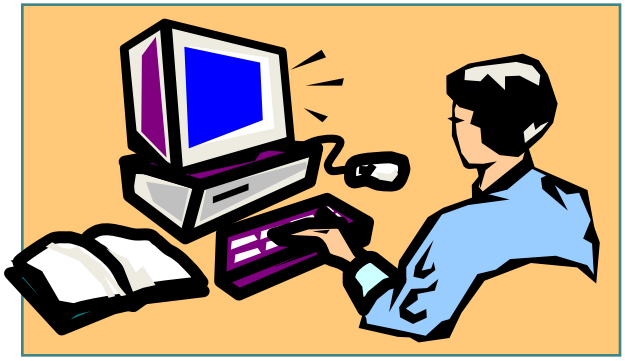
```
void display(struct book tech_book)
{
   printf("Name :%s\n Author :%s\nPages :%d\n", tech_book.name,
                                    tech_book.author,tech_book.pages);

}
```

Output :
Name: Programming in C
Author: Stephen
Pages: 300

#3: Passing Structure Pointers to Function

- **This method involves passing pointers to the structure variables as the function arguments.**
- **In the situations where, more than one member variable is computed in the function, pointers to structures are used.**
- **If the pointer to a structure is passed as an argument to a function, then any changes that are made in the function are visible in the caller.**

Demo - Passing Structure Pointer to Function



Sample Code

```
# include <stdio.h>
# include “str2.h”
# define CURRENT_YEAR 2011
void increment(emprec *x)
{
        if(CURRENT_YEAR - x->birthday.year > 30)
                x->salary += 500;
}
void main(void)
{
        emprec per={“Khan",27,10,62,5500};
        printf(" *** Employee Details ***\n");
```

Sample Code

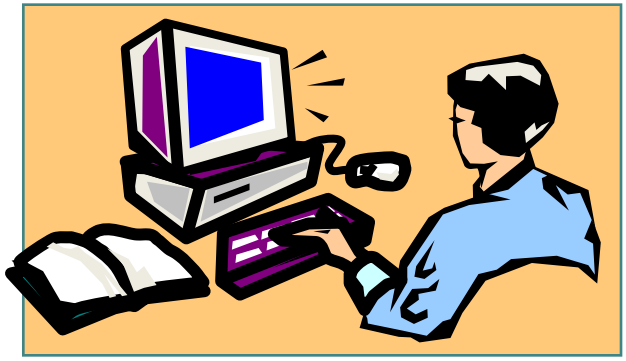
```
        printf("Name :%s \n",per.name);
        printf("Birthdate: %d:%d:%d \n", per.birthday.day,
                per.birthday.month,per.birthday.year);
      printf("Salary :%6.2f\n \n",per.salary);
   /* give increments to employees if age is greater than 30 */
        increment(&per);
        printf(" *** Employee Details ***\n");
        printf("Name :%s \n",per.name);
        printf("Birthdate: %d:%d:%d\n",per.birthday.day,
                        per.birthday.month,per.birthday.year);
        printf("Salary :%6.2f\n",per.salary);
}
```

Structures as Function Values

- **Structures can be returned from functions just as variables of any other type.**
- **Instead of accepting a pointer to a structure, it can construct a structure by itself and return this structure variable.**

Demo on Structure as Function Values

- **Demo on struct_asReturn_type.c program**



Sample Code

```
/* Example- structures and functions */
# include<stdio.h>
struct time
{
    int min,hr,sec;
};
void main(void)
{
   struct time time_udt(struct time);
   struct time or_time,nx_time;
   printf("Enter time(hh:mm:ss):");
   scanf("%d:%d:%d",&or_time.hr,&or_time.min,
                                        &or_time.sec);
```

Sample Code

```
nx_time = time_udt(or_time);
printf("Updted time is :%2d:%2d:%2d\n", nx_time.hr,
                              nx_time.min,nx_time.sec);
}
struct time time_udt(struct time now)
{
   struct time new_time;
   new_time=now;
   ++new_time.sec;
   if(new_time.sec==60)
   {
       new_time.sec=0;
       ++new_time.min;
```

Sample Code

```
if(new_time.min==60)
        {
                new_time.min=0;
                ++new_time.hr;
                if(new_time.hr==24)
                        new_time.hr=0;
         }
   }
   return(new_time);
}
```

Common Best Practices in C Programming

Structures

- **The following program works correctly, but it dumps core after it finishes. So be careful while structure declaration.**

```
struct list
        { char *item; struct list *next; } //missing semicolon

        /* Here is the main program. *
                main(argc, argv) ...
```

  - A missing semicolon causes the compiler to believe that main returns a structure. Since struct-valued functions are usually implemented by adding a hidden return pointer, the generated code for main() tries to accept three arguments, although only two are passed.

Structures

- **Compiler will leave an unnamed, unused hole between members of structure for appropriate alignment.**
  - Use #pragma directive for removing the padding effect .
    Use the following code:
    #pragma pack(push,1)

struct emp
{
char name[3];
char role[2];
double salary;
};
Output: 16 byte
	struct emp
	{
	int empid;
	char band;
	char gender;
	};
	Output: 8 byte

Structures

- Structures cannot be compared as byte-by-byte comparison can be invalidated by random bits present in unused “holes” in the structure
- If you need to compare two structures, you will have to write your own function to do so, field by field.
- Size of report a larger size than expect for a structure type due to padding effect.

Unions

```
union record {
        char *name;
        int refcount : 4;
        unsigned dirty  : 1; //bitfield
    };
```

- **Remember that the colon notation for specifying the size of a field in bits is valid only in structures (an in union); you cannot use this mechanism to specify the size arbitrary variables.**

Typedef

- **The type defined with a typedef is exactly like its counterpart as far as its type declaring power is concerned. However, it cannot be modified like its counterpart**
  - typedef int MYINT
- **Now you can declare an int variable either with:**
  - int a; or MYINT a;
- **However, you cannot declare an unsigned int (using the unsigned modifier) with unsigned MYINT a; although unsigned int a; would be perfectly acceptable**

Typedef

- **typedefs can correctly encode pointer types. Whereas #DEFINES are just replacements done by the preprocessor**

- **For example,**

  typedef char * String_t;

  #define String_d char *

  String_t name, content; String_d subject, grade;

  name, content, and subject are all declared as char *, but grade is declared as a char, which is probably not the intention.

Typedef

- **It is extremely useful to make code more compact and easier to read typedef also allows to declare arrays,**

```
typedef char char_arr[];
char_arr my_arr = "Hello World!\n";
```

- **This is equal to**

```
char my_arr[] = "Hello World!\n";
```

Summary

- **In this lesson, you have learnt:**
  - Individual members cannot be initialized inside the structure declaration.
  - Structures may be passed as function arguments and functions may return structures.
  - Memory from the heap needs to be allocated for a pointer to a structure if you want to store some data. This is done by using malloc() function.
  - Unions like structures, contain members whose individual data types may differ from one another.

Review Question

- **Question 1: Data items that make up a structure can be of different types.**
  - True / False
- **Question 2: The structure variables can be accessed using operator ___**
- **Question 3: It is not possible to create an array of pointer to structures.**
  - True / False

Review Question: Match the Following

1. ->	1. Operator used to access structure element through structure variable
2. .	2. Operator used to assign elements of one structure variable to another structure variable.
3. =	3. Operator used to access structure element through pointer to structure

Lab Session

- **Lab 6**