Rajarata University of Sri Lanka
Department of Physical Sciences

COM1407
Computer Programming

LECTURE 12

STRUCTURES

Piyumi Herath

Objectives

- At the end of this lecture students should be able to;
  - Define, initialize and access C stuctuers.
  - Develop programs using structures in arrays and functions.
  - Use structures within structures and structures as pointers.
  - Apply taught concepts for writing programs.

Introduction

- How do we keep the group of data items that are related but have different data types?
- For example, suppose that along with the game scores, we want to store the name of each player, the country from which they come and their ages.
- Here, the score is a float, the player and the country are character strings and the age is an integer.
- We would like to store these four items together as one unit and to handle and process them together as a group.
- In order to do this we can use structure data type.

Define Structures

- The keyword used to define structure data type is struct.
- We can define a structure as below.

```
struct Participant
{
    char Name[20];
    char country[25];
    float Score;
    int Age;
};
```

Defining a Structure (Cont…)

- *struct* is a keyword and the *Participant* is a name or a tag that we provided.
- All that we are doing is defining a structure called *struct Participant*.
- We are not creating a variable, instead we are creating a new data type that includes or aggregate other C data types.
- You can think of this as a structure template from which structure variables may be defined.
- The order in which the structure members are defined is not important.

Declaring Structure Variables

- The following code shows how structure variables are declared.

```
struct Participant Player1;
struct Participant Player2;
```

or both together

```
struct Participant Player1, Player1;
```

Declaring Structure Variables (Cont…)

```
struct date
{
    int month;  int day;
    int year;
} todaysDate, purchaseDate;
```

- Defines the structure *date and* also declares the variables *todaysDate* and *purchaseDate* to be of this type.

Initializing Structure Variables

- The following code shows how you can initialize structure variables during declaration.

Eg 1:

```
struct Participant Player1= {"Marianne", "USA", 4.6, 20};
```

Eg. 2:

```
struct Participant Player 1, Player2 = {"Anne", "Germany", 9.4, 25};

//In 2nd example we are only initializing one struct variable.
```

Initializing Structure Variables (Cont…)

- Also, below initialization is also valid.

```
struct date
{
    int month;
    int day;
    int year;
} todaysDate = { 1, 11, 2005 };
```

- This example defines the structure *date* and initialize *todaysDate* with values as indicated.

Accessing Values

- With structures, we must follow the variable name by a **period**, followed by the member name to access one member.
- This operator is known as member access operator (.).
- Example,

  struct Participant Player2 = {"Marianne", "USA", 4.6, 20};

  Player2.score is equal to 4.60.

  printf("Score : %f ", **Player2.score**);

Accessing Values (Cont…)

```
#include <stdio.h>

struct Participant
{
      char name[20];
      char country[25];
      float score;  int Age;
};
int main ()
{
       struct Participant Player1, Player2= {“Kumar”, “India”, 4.6, 19};
       printf ("%s\n", Player2.name);
       printf ("%i\n", Player2.age);
 return 0;
}
```

Accessing Values (Cont…)

```
#include <stdio.h>

struct Participant

{
        char name[20];
        char country[25];
        float score;
        int age;

};

int main (){
        struct Participant Player1 = {"Sugath", "Sri Lanka",5.5, 20};
        struct Participant Player2 = {“Kumar", “India", 4.6, 19};

        printf ("%s : %s\n", Player1.name, Player1.country);

        printf ("%s : %s\n", Player2.name, Player2.country);

return 0;

}
```

Exercise

Write a program to store day, month and year into a structure. Print "Wish you a happy new year " if both month and day of a particular day is equal to 1

Exercise

```
#include <stdio.h>
struct Day
{     int year;
      int month;
      int day;
};
int main (){
        struct Day today;
        today.month = 1;
        today.day = 1;
        today. year= 2017;
        if (today.month == 1 && today.day == 1){
                printf ("Wish you a happy new year");
        }
return 0; }
```

Array of Structures

- C does not limit you to storing simple data types inside an array; it is perfectly valid to define an *array of structures*.
- For example,

```
struct date birthdays[15];
```

defines the array *birthdays* to contain 15 elements of type *struct date*.

Array of Structures (Cont…)

```
struct Participant Player[2];
```

- This will create an array called Player[ ] with only 2 elements, where each element is a structure of type *struct Participant*.

- **Initializing Player array**

struct Participant Player[2]= { {“Kumar", “India", 4.6, 19},
{“Mary", "Australia", 4.8, 20} };

Array of Structures (Cont…)

```
#include <stdio.h>
struct Participant {
        char Name[20];
        char Country[25];
         float Score;
        int Age;
};
int main ()
{       struct Participant Player[2] = { {"Sugath", "Sri Lanka",5.5, 20},
                                         {"Kumar", "India", 4.6, 19}};
        printf ("%s : %s\n", Player[0].Name, Player[0].Country);
        printf ("%s : %s\n", Player[1].Name, Player[1].Country);
        return 0;
}
```

Array of Structures (Cont…)

```
#include <stdio.h>

struct time
{
        int hour;
        int min;
        int sec;
};

int main ()
{
      struct time runTime1 [5] ={ {12, 10, 15}, {12, 30, 0}, {13, 15, 0} };

      printf ("%i\n",runTime1[0].hour);
      printf ("%i\n",runTime1[0].min);
      printf ("%i\n",runTime1[0].sec);

       return 0;
}
```

Structures Containing Structures

- A structure can be nested inside another structure. In other words, the members of a structure can be of type structure too.

```
#include <stdio.h>

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

struct Employee{
        char ename[20];
        int ssn;
        float salary;
        struct date doj;
} emp = {"Pritesh",1000,1000.50,{22,6,1990}};
```

```
int main()
{
        printf("\nEmployeeName          : %s",emp.ename);
        printf("\nEmployee SSN    : %d",emp.ssn);
        printf("\nEmployee Salary : %f ",emp.salary);

        printf("\nEmployee DOJ  : %d/%d/%d",
        emp.doj.date,emp.doj.month,emp.doj.year);
return 0;
}
```

Pointer to structure

- Define pointers to structures in similar way as you define pointer to any other variable as follows:

  struct student *s_pointer;

- To store the address of a structure variable in the above defined pointer variable, we place the & operator before the structure's name as follows:

  s_pointer = &s1; // Assume **s1** is of type student

- To access the members of a structure using a pointer to that structure, you must use the **->** arrow operator as follows:

  s_pointer->name;

Passing Structures to functions

- You can pass a structure as a function argument in similar way as you pass any other variable or pointer.

Pass structure by Value

```
#include <stdio.h>
struct student {
        int index;
       char name[50];
};
void display( struct student s);

int main() {
   struct student s1;

   printf("Enter Index number: ");
   scanf("%d", &s1.index);
   printf("Enter name: ");
   scanf("%s", s1.name);
    display(s1);
   return 0;
}
void display( struct student s) {
       printf("\n Displaying information\n--------------\n");
       printf("\n Index Number: %d\n", s.index);
       printf(" Name        : %s\n", s.name);
}
```

```
Enter Index number: 3891
Enter name: Nayomi

Displaying information
--------------

Index Number: 3891
Name         : Nayomi
```

Pass structure by Reference/Pointer

```
#include <stdio.h>
struct student {
   int index;
   char name[50];
};
void display(struct student *s);
int main() {
   struct student s1;

   printf("Enter Index number: ");
   scanf("%d", &s1.index);
   printf("Enter name: ");
   scanf("%s", s1.name);
   display(&s1);
   return 0;
}
void display(struct student *s) {
      printf("\n Displaying information\n----------\n");
      printf("\n Index Number: %d\n", s->index);
      printf(" Name        : %s\n", s->name);
}
```

We can pass structure as a pointer too.
For that we need to
create a pointer to the structure
Eg:
struct student *std_ptr;
std_ptr = &s;
Then call function as
display(std_ptr);

Note that instead of the (.)operator here we are using **->** operator to assign/access pointer structure variables

Thank You