SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A’ Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit V – STRUCTURE AND UNION

Topic : Structures within Structures - Union

Topics Covered

- **Structures and Unions:**
  - » **Structure Initialization**
  - » **Array of Structures**
  - » **Array within Structures**
  - » **Structures with Structures**
  - » **Unions**

Structure Initialization

- The initialization must be done only in the declaration of the actual variables.
- Compile time initialization of a structure variable have the following elements.

1.The keyword struct.

2. The structure tag name.

3.The name of a variable to be declared.

Structure Initialization

4. The assignment operator =
5. A set of values for the members of a structure variable, separated by commas and enclosed in the braces.
6. A terminating semicolon.

Structure Initialization

- **Another method for structure initialization:**

```
 main()
 {
 struct student
{
 int rollno;
 float marks;
 };
 struct student student 1= {101, 90};
 struct student student 2 = { 102, 80};
```

Structure Initialization

RULES:

- We cannot initialize individual member inside the structure template.
- It is permitted to have a partial initialization.
- Here we can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.

Structure Initialization

- The **uninitialized members** will be **assigned default values** as follows.
  - **Zero** for integer and floating point numbers
  - **‘\0’** for characters and strings.

Copying and Comparing Structure Variables

- Two variables of the same structure type can be copied the same way as ordinary variables.
- Example:

```
person1 = person2;

person2 = person1;
```

- C **does not permit** any **logical operation on structure variables.**
- In case we need to compare them, we may do so by **comparing members individually**.

Operations on individual members

- The individual members are identified using the **member operator**, the dot.
- A member with the dot operator along with its structure variable can be treated like any other variable name and therefore can be **manipulated using expressions and operators.**

Operations on individual members

- **Apply increment and decrement operators to numeric type members**.
- The **precedence** of the member operator is **higher than all arithmetic and relational operators** and therefore **no parentheses are required.**

Array of Structures

- In C Programming, structures are useful to group different data types to organize the data in a structural way.
- And arrays are used to group the same data type values.
- For example, to store employee details such as name, id, age, address, and salary.
- We usually group them as employee structure with the members mentioned above.
- We can create the structure variable to access or modify the structure members

Array of Structures

Syntax

```
struct struct-name
 {
 datatype var1;
datatype var2;
 - - - - - - - - - - - - - - - - - - - - - datatype varN;
 };
struct struct-name obj [ size ];
```

Array of Structures

```
/* Array of Structures in C Initialization */
struct Employee
 {
 int age;
char name[50];
 int salary;
}
Employees[4] = { {25, "Suresh", 25000}, {24, "Tutorial", 28000},
    {22, "Gateway", 35000}, {27, "Mike", 20000} };
```

Array of Structures

```
#include<stdio.h>
struct Point
{
int x, y;
};
int main()
{
// Create an array of structures
struct Point arr[10];
// Access array members
arr[0].x = 10;
arr[0].y = 20;
printf("%d %d", arr[0].x, arr[0].y);
return 0;
}
```

Array of Structures

```
#include<stdio.h>
struct Employee
{
char ename[10];
int sal;
};
struct Employee emp[5];
int i, j;
void ask()
{
for(i = 0; i < 3; i++)
{
printf("\nEnter %dst Employee record:\n", i+1);
printf("\nEmployee name:\t");
scanf("%s", emp[i].ename);
printf("\nEnter Salary:\t");
scanf("%d", &emp[i].sal);
}
```

```
printf("\nDisplaying Employee
record:\n");
for(i = 0; i < 3; i++)
{
printf("\nEmployee name is %s",
emp[i].ename);
printf("\nSalary is %d",
emp[i].sal);
}
}
int main()
{
ask();
}
```

Array within Structures

- Sometimes, arrays may be the member within structure, this is known as arrays within structure.
- Accessing arrays within structure is similar to accessing other members
- **<u>Purpose of Array within Structure</u>**
- When we want to store a string value, then we have to go for array within structure.
- Because name comes under character data type alone, thus array is capable of storing data of same data type.

Array within Structures

- **Syntax**

```
struct struct-name
{

datatype var1; // normal variable
datatype array [size]; // array variable
----------------
----------------
datatype varN;
 };
struct struct-name obj;
```

Array within Structures

```
#include <stdio.h>
int main()
{
struct student {
char name[30];
int rollno;
} stud;
printf ("Enter your RollNo : ");
scanf ("%d",&stud.rollno);
printf ("\nEnter your Name : ");
scanf ("%s", stud.name);
printf ("\nRollNo : %d\n Name : %s", stud.rollno, stud.name);
return 0;
}
```

Array within Structures

```
#include <stdio.h>
int i;
struct student {
char name[30];
int rollno;
} stud[3];
int main()
{
for(i=0; i<3; i++)
{
printf ("\nEnter your RollNo : ");
scanf ("%d",&stud[i].rollno);
printf ("\nEnter your Name : ");
scanf ("%s", stud[i].name);
}
printf("\nList of all records");
for (i=0; i<3; i++)
{
 printf ("\nRollNo : %d\n Name : %s", stud[i].rollno, stud[i].name);
}
return 0; }
```

Array within Structures

```
#include<stdio.h>
struct Student
{
int Roll;
char Name[25];
int Marks[3]; //Statement 1 : array of marks
int Total;
float Avg;
};

int main()
{
int i;
struct Student S;

printf("\n\nEnter Student Roll : ");
scanf("%d",&S.Roll);
```

```
printf("\n\nEnter Student Name : ");
scanf("%s",S.Name);
S.Total = 0;
for(i=0;i<3;i++)
{
printf("\n\nEnter Marks %d : ",i+1);
scanf("%d",&S.Marks[i]);

S.Total = S.Total + S.Marks[i];
}

S.Avg = S.Total / 3;
printf("\nRoll : %d",S.Roll);
printf("\nName : %s",S.Name);
printf("\nTotal : %d",S.Total);
printf("\nAverage : %f",S.Avg);
}
```

Structures within Structures

- In C, a structure declaration can be placed inside another structure. This is also known as **nesting of structure.**
- The declaration is same as the declaration of data type in structure.
- **Structure within structure (or) nesting of structure is used to create complex records.**
- There are two methods to declare a structure within structure. Programmers can use either one method to declare structure within structure.
  - Embedded Structure Declaration
  - Two Separate Structure Declaration

Structures within Structures

- When a **structure contains another structure, it is called nested structure.**
- For example, two structures named Address and Employee.
- To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.

Structures within Structures

- Syntax

```
struct structure1
 {
  - - - - - - - - - - - - - - - - - - - -


};
struct structure2
{
 - - - - - - - - - - - - - - - - - - - - -


struct structure1 obj;


};
```

Structure within Structure

```
#include <stdio.h>
int main()
{
struct student  {
char name[30];
  struct avg  {
    int sub1, sub2, sub3;
    float average;
    }avg1;
};
struct student stud1;
printf("Enter the Name of the student ");
scanf("%s", stud1.name);
printf("\nEnter the marks of the student ");
scanf("%d %d %d ", &stud1.avg1.sub1, &stud1.avg1.sub2, &stud1.avg1.sub3);
stud1.avg1.average = (stud1.avg1.sub1 + stud1.avg1.sub2 + stud1.avg1.sub3)/3;
printf("\n-------Student Details-------\n ");
printf("%s", stud1.name);
printf("\nsub1: %d \n sub2: %d \n sub3: %d ", stud1.avg1.sub1, stud1.avg1.sub2,
stud1.avg1.sub3);
printf("\n Average: %f ", stud1.avg1.average);
return 0;
}
```

Structures with Structures

Note:

- The above program uses Embedded type declaration. Structure avg is defined within the structure student.

Structures with Structures

```
#include <stdio.h>
int main()
{
struct avg{
int sub1, sub2, sub3;
float average;
}avg1;
struct student{
char name[30];
struct avg avg1;
};
struct student stud1;
printf("Enter the Name of the student ");
scanf("%s", stud1.name);
printf("\nEnter the marks of the student ");
scanf("%d %d %d ", &stud1.avg1.sub1, &stud1.avg1.sub2, &stud1.avg1.sub3);
stud1.avg1.average = (stud1.avg1.sub1 + stud1.avg1.sub2 + stud1.avg1.sub3)/3;
printf("\n-------Student Details-------\n ");
printf("%s",stud1.name);
printf("\nsub1 : %d \n sub2 : %d \n sub3 : %d ",stud1.avg1.sub1, stud1.avg1.sub2,
stud1.avg1.sub3);
printf("\nAverage : %f %", stud1.avg1.average);
return 0;
}
```

Structures with Structures

Note:

- The above program uses two structure declaration method.
- Structure avg is defined outside the structure student.

Structures with Structures

```
#include<stdio.h>

struct Address
{
char HouseNo[25];
char City[25];
char PinCode[25];
};

struct Employee
{
int Id;
char Name[25];
float Salary;
struct Address Add;
};

int main()
{
int i;
struct Employee E;

printf("\n\tEnter Employee Id : ");
scanf("%d",&E.Id);

printf("\n\tEnter Employee Name : ");
scanf("%s",E.Name);
```

```
printf("\n\tEnter Employee Salary : ");
scanf("%f",&E.Salary);

printf("\n\tEnter Employee House No : ");
scanf("%s",E.Add.HouseNo);

printf("\n\tEnter Employee City : ");
scanf("%s",E.Add.City);

printf("\n\tEnter Employee House No : ");
scanf("%s",E.Add.PinCode);

printf("\nDetails of Employees");
printf("\n\tEmployee Id : %d",E.Id);
printf("\n\tEmployee Name : %s",E.Name);
printf("\n\tEmployee Salary : %f",E.Salary);
printf("\n\tEmployee House No :
%s",E.Add.HouseNo);
printf("\n\tEmployee City : %s",E.Add.City);
printf("\n\tEmployee House No :
%s",E.Add.PinCode);
}
```

UNION

- A union is a user-defined type similar to structures in c except for one key difference.
- Structure allocate enough space to store all its members whereas unions allocate the space to store only the largest member.

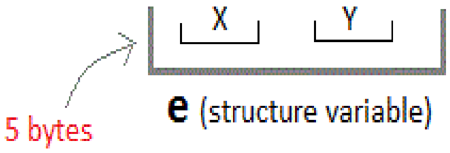
UNION

- **Syntax – Union**

```
union unionname
 {
data type member1;
 data type member2;
 } variable1, variable2;
```

Structure

```
struct Emp
{
 char X ;     // size 1 byte
 float Y ;    // size 4 byte
} e ;
```



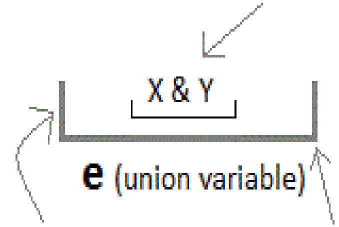
5 bytes

e (structure variable)

Unions

```
union Emp
{
 char X ;
 float Y ;
} e ;
```

Memory Sharing



e (union variable)

4 bytes

allocates storage
equal to largest one

Difference Between Structure and Union

```
#include <stdio.h>
int main()
{
//structures declartion
struct sample{
double d1; //occupies 8 bytes in memory
float f1; //occupies 4 bytes in memory
}s1;
//Union declartion
union samp{
double d2; //occupies 8 bytes in memory
float f2; //occupies 4 bytes in memory
}u1;
printf("\nSize of Structure : %ld ",sizeof(s1));
printf("\nSize of Union : %ld",sizeof(u1));
return 0;
}
```

UNION

- **Accessing a Union Member**
- A union member can be accessed similar to structure member, that by using (.)dot or period operator.
- The general format is as follows,

unionvariablename.membername;

UNION

```
#include <stdio.h>
int main()
{
//structures declartion
struct sample{
int a;
int b;
}s1;
//Union declartion
union samp{
int c;
int d;
}u1;
s1.a = 10;
s1.b = 20;
printf("\nThe Structure member values a and b are : %d %d ", s1. a, s1.b);
u1.c = 30;
u1.d = 40;
printf("\nThe Union member values c and d are : %d %d ", u1. c, u1.d);
return 0;
}
```

UNION

```
#include <stdio.h>
#include <string.h>
union Data {
int i;
float f;
char str[20];
};
int main( ) {

union Data data;

data.i = 10;
data.f = 220.5;
strcpy( data.str, "C Programming");

printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str);

return 0;
}
```

UNION

- When the above code is compiled and executed, it produces the following result:

```
data.i : 1917853763
data.f4122360580327794860452759994368.000000
data.str : C Programming
```

- Here, the values of **i** and **f** members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of **str** member is getting printed very well.

UNION

```
#include <stdio.h>
#include <string.h>
union Data {
int i;
float f;
char str[20];
};
int main( ) {

union Data data;

data.i = 10;
printf( "data.i : %d\n", data.i);
data.f = 220.5;
printf( "data.f : %f\n", data.f);
strcpy( data.str, "C Programming");
printf( "data.str : %s\n", data.str);

return 0;
}
```

Output:

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

Summary

- A structure variable declaration is similar to the declaration of variables of any other data type.
- The initialization must be done only in the declaration of the actual variables.
- We can access and assign values to the members of a structure in a many different ways.
- A union is a user-defined type similar to structures in c except for it allocate the space to store only the largest member.

Thank you