**Structure & Union**

**Structure**: - A structure is a user defined data type. It is a collection of data items or variables of different data types that are referenced under the same name. It is a convenient way of keeping related information together. All the elements of a structure are stored at contiguous memory locations.

A variable of structure type can store multiple data items of different data types under the one name.

**Defining of Structure**

The syntax of defining a structure is

struct <struct\_name>

{

<data\_type> <variable\_name>;

<data\_type> <variable\_name>;

……..

<data\_type> <variable\_name>;

};

**Declaring a Structure Variable**

A structure has to declared, after the body of structure has defined. The syntax of declaring a structure is

struct <struct\_name> <variable\_name>;

**Example**:- The data of employee in company that has name, Employee ID, salary, address, phone number. This information can be stored in structure data type.

**Step 1: The structure of Employee is declared as**

struct employee

{

int emp\_id;

char name[20];

float salary; member variable

char address[50]; or

int dept\_no; structure member

int age;

};

**Step 2: Syn**tax to declare the variable for structure “employee”

**struct employee e1;**

Here e1 variable contains 6 members that are defined in structure.

Putting together step 1 and step 2

struct employee{ struct employee{

int emp\_id; 2 bytes int emp\_id;

char name[20]; 20Bytes char name[20];

float salary; 4Bytes float salary;

char address[50]; 50 Bytes char address[50];

int dept\_no; 2 Bytes int dept\_no;

int age; 2 Bytes int age;

}; }e1;

**struct employee e1;**

We can declare structure and the structure variable using any one of the above syntax.

|  |
| --- |
| emp\_id |
| name[20] |
| salary |
| address[50] |
|  |
| age |

**Memory Space Allocation :-** 80 bytes will be allocated to the structure variable e1.

8000

8002

8022

8026

8076

8078

8079 employee e1

**Initializing and accessing a Structure Members**

**Compile time Initialization**:- this can be don in two ways

* The members of individual structure variable is initialize one by one using a **dot(.)** operator. The Dot operator is preceded by Structure variable and succeeded by member variable.

Syntax

**< Structure variable > . < member variable>**

Example

**e1.emp\_id=1;**

**e1.dept\_no=1**

**e1.age=35;**

**e1.salary=12000;**

**e1.address=“3 vikas colony new delhi”;**

* The members of individual structure variable are in a single statement.

**struct employee e1 = {1, “Hemant”,12000, “3 vikas colony new delhi”,10, 35};**

**Run time Initialization: -** The members of individual structure variable is initialize initialized in run time using scanf() and Dot operator. We can explain it with help of following example.

**write a program in C to enter the employee information and display it.**

**#include <stdio.h>**

**#include <conio.h>**

**struct employee**

**{**

**int emp\_id;**

**char name[20];**

**float salary;**

**char address[50];**

**int dept\_no;**

**int age;**

**};**

**void main ( )**

**{ struct employee e1;**

**printf (“Enter the employee id \t name\t Salary\t Address\tDepartment \t age of the employee 1\n ”);**

**scanf(“%d%s%f%s%d%d”,&e1.emp\_id,e1.name,&e1.salary,e1.address,&e1.dept\_no, ,&e1.age);**

**printf (“details of the employee 1\n ”);**

**printf(“emp\_id=%d\nname=%s\nSalary=%d\nAddress=%s\nDept=%d\nAge=%d”,e1.emp\_id, e1.name,e1.salary, e1.address, e1.dept\_no);**

**}**

**Array of Structure**

The array of structure is used to store the large number of similar records. For example to store the record of 100 employees then array of structure is used. The method to define and access the array element of array of structure is similar to other array. The syntax to define the array of structure is

Struct <struct\_name> <var\_name> <array\_name> [<value>];

For Example:-

Struct employee e[100];

Representation

**e[0] e[1] e[2]………………………….e[99]**

**emp1 emp2 emp3 emp100**

**Program to implement the Array of Structure**

#include <stdio.h>

#include <conio.h>

struct employee

{

int emp\_id;

char name[20];

float salary;

char address[50];

int dept\_no;

int age;

};

void main ( )

{

struct employee e[5];

int i;

for (i=1; i<=5; i++)

{

printf (“Enter the employee id of employee”);

scanf (“%d”,&e[i].emp\_id);

printf (“Enter the name of employee”);

scanf (“%s”,e[i].name);

printf (“Enter the salary of employee”);

scanf (“%f”,&e[i].salary);

printf (“Enter the address of employee”);

scanf (“%s”, e[i].address);

printf (“Enter the department of employee”);

scanf (“%d”,&e[i].dept\_no);

printf (“Enter the age of employee”);

scanf (“%d”,&e[i].age);

}

for (i=1; i<=5; i++)

{

printf (“The employee id of employee is : %d”, e[i].emp\_id);

printf (“The name of employee is: %s”,e[i].name);

printf (“The salary of employee is: %f”, e[i].salary);

printf (“The address of employee is : %s”, e[i].address);

printf (“The department of employee is : %d”,e[i].dept\_no);

printf (“The age of employee is : %d”, e[i].age);

}

getch();

}

**Differentiate between array and structure: -**

|  |  |
| --- | --- |
| Arrays | Structures |
| 1. An array is a collection of related data elements of same type. | 1. Structure can have elements of different  types. |
| 2. An array is a derived data type | 2. A structure is a user-defined data type |
| 3. Any array behaves like a built-in data types. We have to declare an array variable and use it. | 3. But in the case of structure, first we have to design and declare a data structure before the variable of that type are declared and used |
| 4. Array allocates static memory and uses index / subscript for accessing elements of the array. | 4. Structures allocate dynamic memory and uses (.) operator for accessing the member of a structure. |
| 5. Array element access takes less time in comparison with structures | 5. Structure element access takes more time in comparison with array |

**Union: -** A union is a user defined data type like structure. It is a collection of data items or variables of different data types that are referenced under the same name. It is a convenient way of keeping related information together.

The union data type allocates the space equal to space need to hold the largest data member of union. The union allows different types of variable to share same space in memory. There is no other difference between structure and union than internal difference. The method to declare, use and access the union is same as structure.

**Defining of Union**

**The syntax of defining a structure is**

union <union\_name>

{

<data\_type> <variable\_name>;

<data\_type> <variable\_name>;

……..

<data\_type> <variable\_name>;};

**The union of Employee is declared as**

union employee

{

int emp\_id;

char name[20];

float salary; member of union

char address[50]; or

int dept\_no; member variable

int age;

} u1,u2; union Variable

**Initializing and accessing a union Members :- initialization of union members are same as Structure members using Dot(.)operator**

**Memory Space Allocation :-** 50 bytes will be allocated to the union variable u1. 50 byte is equal to space need to hold the largest data member of union.

#include<stdio.h>

#include<conio.h>

union abc{

int i;

char ch;

};

Void main()

{

union abc u;

u.i= 67;

printf(“value of char variable is %c”,u.ch);

printf(“value of int variable is %d”,u.i);

getch();

}

ouput:-

value of char variable is C

value of char variable is 67

**Memory Diagram: -**

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **1** |

char ch

int i

|  |  |
| --- | --- |
| **Structure** | **Union** |
| 1.The keyword  **struct** is used to define a structure | 1. The keyword union is used to define a union. |
| 2. When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is equal to the sum of sizes of its members | 2. When a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of union is equal to the size of largest member. |
| 3. Each member within a structure is assigned unique storage area of location. | 3. Memory allocated is shared by individual members of union. |
| 4 Altering the value of a member will not affect other members of the structure. | 4. Altering the value of any of the member will alter other member values. |
| 5. Individual member can be accessed at a time | 5. Only one member can be accessed at a time. |
| 6. Several members of a structure can initialize at once. | 6. Only the first member of a union can be initialized. |
| 7. Example  **struct stud**  **{**  **char name[15];**  **int roll\_no;**  **int batch;**  **} ;**  Size of this structure is 19 bytes (sum of sizes of its members). | 7.Examle  **union stud**  **{**  **char name[15];**  **int roll\_no;**  **int batch;**  **} ;**  Size of this structure is 15 bytes (size of largest member). |