**Chapter -8**

**Structure and Union**

**Introduction**

A structure is a collection of logically related data items grouped together under a single name. In structure the individual elements may differ in type, that’s why we can regard structure as a heterogeneous user-defined data type. The data items enclosed within a structure are known as members. A structure can be considered as a template used for defining a collection of variables under a single name. Structures help programmers to group elements of different data types into a single logical unit (Unlike arrays which permit a programmer to group only elements of same data type).

Syntax for structure definition is:

struct struct\_name

{

data\_type mem1;

data\_type mem2;

…………………

data\_type memn;

};

The structure definition starts with keyword *struct* followed by an identifier which is a tag name. The tag name is structure name and can be used for instantiating structure variable. struct\_name is referred to as structure name or structure tag name, and mem1, mem2, memn are known as structure members.

After the structure has been specified, the structure variable can be declared as standard data type:

struct struct\_name variable\_name;

Where *struct* is a required keyword, struct\_name is the name that appeared in the structure definition and variable\_name is structure variable of type struct\_name.

Suppose we want to store a date inside a C program. Then, we can define a structure called date with three elements day, month and year. The syntax of this structure is as follows:

struct date

{

int day;

int month;

int year;

};

Example 2

struct student

{

char name[15];

int roll;

float fee;

};

struct student st;

A structure type is usually defined at the beginning of a program. This usually occurs just after the main() statement in a file. Then a variable of this structure type is declared and used in the program. For example:

struct date order\_date;

Note : When you first define a structure in a file, the statement simply tells the C compiler that a structure exists, but causes no memory allocation. Only when a structure variable is declared, memory allocation takes place.

**Accessing members of structure**

For accessing any member of a structure variable, we use the dot(.) operator which is also known as the period or membership operator. Syntax:

structvariable\_name.member

**Initialization of structure Variables**

The syntax of initializing structure variables is similar to that of arrays. All the values are given in curly braces and the number, order and type of these values should be same as in the structure template definition.

Example

struct student

{

char name[15];

int roll;

float fee;

};

struct student st={“Sonia”,23,1450.50};

**Declaring and Initializing Multiple Variables**

struct student

{

char name[20];

int roll;

float marks;

}

std1 = {"Pritesh",67,78.3};

std2 = {"Don",62,71.3};

In this example, we have declared two structure variables in above code. After declaration of variable

we have initialized two variable.

std1 = {"Pritesh",67,78.3};

std2 = {"Don",62,71.3};

**Initializing Single member**

struct student

{

int mark1;

int mark2;

int mark3;

} sub1={67};

Though there are three members of structure,only one is initialized , Then remaining two members are initialized with Zero.

**Initializing inside main**

struct student

{

int mark1;

int mark2;

int mark3;

};

void main()

{

struct student s1 = {89,54,65};

- - - - -

- - - - -

- - - - -

};

When we declare a structure then memory won’t be allocated for the structure. i.e only writing below declaration statement will never allocate memory

struct student

{

int mark1;

int mark2;

int mark3;

};

We need to initialize structure variable to allocate some memory to the structure.

struct student s1 = {89,54,65};

**Declaring Structure Variable**

In C we can group some of the user defined or primitive data types together and form another compact way of storing complicated information is called as Structure. Let us see how to declare structure in c programming language –

Syntax Of Structure in C Programming :

struct tag

{

data\_type1 member1;

data\_type2 member2;

data\_type3 member3;

};

Structure Alternate Syntax :

struct <structure\_name>

{

structure\_Element1;

structure\_Element2;

structure\_Element3;

...

...

};

Some Important Points Regarding Structure in C Programming :

1. Struct keyword is used to declare structure.

2. Members of structure are enclosed within opening and closing braces.

3. Declaration of Structure reserves no space.

4. It is nothing but the “ Template / Map / Shape ” of the structure .

5. Memory is created , very first time when the variable is created /Instance is created.

Different Ways of Declaring Structure Variable :

Immediately after Structure Template

struct date

{

int date;

char month[20];

int year;

}today;

// 'today' is name of Structure variable

Declare Variables using struct Keyword

struct date

{

int date;

char month[20];

int year;

};

struct date today;

where “date”is name of structure and “today” is name of variable.

Declaring Multiple Structure Variables

struct Book

{

int pages;

char name[20];

int year;

}book1,book2,book3;

We can declare multiple variables separated by comma directly after closing curly.

Write a program to read name, rollno and fee of a student and display it on screen using structure.

#include<stdio.h>

#include<conio.h>

void main()

{

struct student

{

char name[15];

int roll;

float fee;

};

struct student st;

clrscr();

printf("Enter name rollno and fee of a student\n");

scanf("%s%d%f",st.name,&st.roll,&st.fee);

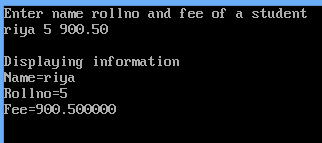
printf("\nDisplaying information\n");

printf("Name=%s \nRollno=%d \nFee=%f",st.name,st.roll,st.fee);

getch();

}

**Output**



**Array of structures**

We can declare array of structures where each element of array is of structure type. Structure is collection of different data type. An object of structure represents a single record in memory, if we want more than one record of structure type, we have to create an array of structure or object. As we know, an array is a collection of similar type, therefore an array can be of structure type.

Syntax for declaring structure array

struct struct-name

{

datatype var1;

datatype var2;

- - - - - - - - -

- - - - - - - - -

datatype varN;

};

struct struct-name obj [ size ];

Example

#include<stdio.h>

struct Employee

{

int Id;

char Name[25];

int Age;

long Salary;

};

void main()

{

int i;

struct Employee Emp[ 3 ]; //Statement 1

for(i=0;i<3;i++)

{

printf("\nEnter details of %d Employee",i+1);

printf("\n\tEnter Employee Id : ");

scanf("%d",&Emp[i].Id);

printf("\n\tEnter Employee Name : ");

scanf("%s",&Emp[i].Name);

printf("\n\tEnter Employee Age : ");

scanf("%d",&Emp[i].Age);

printf("\n\tEnter Employee Salary : ");

scanf("%ld",&Emp[i].Salary);

}

printf("\nDetails of Employees");

for(i=0;i<3;i++)

printf("\n%d\t%s\t%d\t%ld",Emp[i].Id,Emp[i].Name,Emp[i].Age,Emp[i].Salary);

}

Output :

Enter details of 1 Employee

Enter Employee Id : 101

Enter Employee Name : Suresh

Enter Employee Age : 29

Enter Employee Salary : 45000

Enter details of 2 Employee

Enter Employee Id : 102

Enter Employee Name : Mukesh

Enter Employee Age : 31

Enter Employee Salary : 51000

Enter details of 3 Employee

Enter Employee Id : 103

Enter Employee Name : Ramesh

Enter Employee Age : 28

Enter Employee Salary : 47000

Details of Employees

101 Suresh 29 45000

102 Mukesh 31 51000

103 Ramesh 28 47000

In the above example, we are getting and displaying the data of 3 employee using array of object. Statement 1 is creating an array of Employee Emp to store the records of 3 employees.

* Define a structure containing members as roll no, name, course and semester and write program to input information about ‘n’ students and display the name and course of all students.

#include<stdio.h>

#include<conio.h>

void main()

{

struct student

{

int roll;

char name[15];

char course[15];

char sem[15];

};

struct student st[100];

int i,n;

clrscr();

printf("\nHow many students are there: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n Enter rollno, name, course and semester: ");

scanf("%d%s%s%s",&st[i].roll,st[i].name,st[i].course,st[i].sem);

}

printf("\nName\tCourse");

for(i=0;i<n;i++)

{

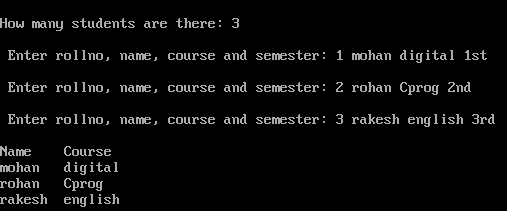
printf("\n%s\t%s",st[i].name,st[i].course);

}

getch();

}

Output



**Arrays within structure**

We can have an array as a member of structure.

Syntax for array within structure

struct struct-name {

datatype var1; // normal variable

datatype array [size]; // array variable

- - - - - - - - -

- - - - - - - - -

datatype varN;

};

struct struct-name obj;

Example

* Write a program to read rollno, name and marks of students in 5 different subjects for ‘n’ students and display all records of students in appropriate format.

#include<stdio.h>

#include<conio.h>

void main()

{

struct student

{

int roll;

char name[15];

int submarks[5];

};

struct student st[100];

int i,j,n;

clrscr();

printf("\nHow many students are there: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n Enter rollno, name: ");

scanf("%d%s",&st[i].roll,st[i].name);

for(j=0;j<5;j++)

{

printf("\n Enter subject marks for %s: ",st[i].name);

scanf("%d",&st[i].submarks[j]);

}

}

printf("\nRollno\tName\tsub1\tsub2\tsub3\tsub4\tsub5");

for(i=0;i<n;i++)

{

printf("\n%d\t%s",st[i].roll,st[i].name);

for(j=0;j<5;j++)

{

printf("\t%d",st[i].submarks[j]);

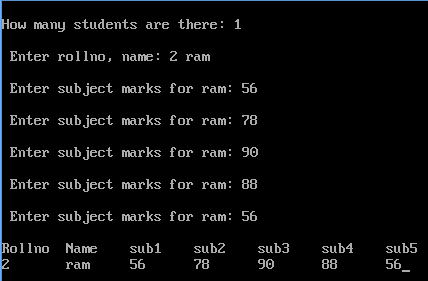
}

}

getch();

}

Output



**Nested structure**

The members of a structure can be of any data type including another structure type i.e we can include a structure within another structure. A structure variable can be a member of another structure. This is called nesting of structure.

Syntax:

struct struct\_name1

{

data\_type mem1;

data\_type mem2;

…………………

data\_type memn;

};

struct struct\_name2

{

data\_type mem1;

data\_type mem2;

…………………

struct\_name1 structure\_variable1;

data\_type memn;

};

struct\_name2 structure\_variable2;

Example

* Define a structure **date** having integer members to store day, month and year. Define another structure **student** having members as rollno, name and date\_of\_birth. Now write a program to accept and display information about ‘n’ students.

#include<stdio.h>

#include<conio.h>

void main()

{

struct date

{

int year;

int month;

int day;

};

struct student

{

int roll;

char name[15];

struct date dob;

};

struct student st[100];

int i,n;

clrscr();

printf("\n How many students are there: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n Enter rollno,name and date of birth in (year-month-day): ");

scanf("%d%s%d%d%d",&st[i].roll,st[i].name,&st[i].dob.year,&st[i].dob.month,&st[i].dob.day);

}

printf("\nRollno\tName\tDOB(Year-Month-Day)");

for(i=0;i<n;i++)

{

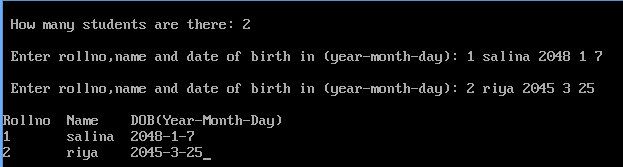
printf("\n%d\t%s\t%d-%d- %d",st[i].roll,st[i].name,st[i].dob.year,st[i].dob.month,st[i].dob.day);

}

getch();

}

Output



**Pointer to structure**

We can have pointer to structure, which can point to the starting address of a structure variable. These pointers are called structure pointers. While accessing structure members through pointers we have to use arrow operator (->) which is formed by hyphen symbol and greater than symbol.

Example

#include<stdio.h>

#include<conio.h>

void main()

{

struct customer

{

int id;

char name[15];

char address[15];

};

struct customer cu={1,"Binod","Bharatpur"};

struct customer \*ptr;

ptr=&cu;

clrscr();

printf("\n ID :%d",ptr->id);

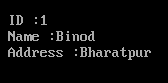
printf("\n Name :%s",ptr->name);

printf("\n Address :%s",ptr->address);

getch();

}

Output



**Passing structure to function**

We can pass whole structure as an argument to a function. In C, structure can be passed to functions by two methods:

1. Passing by value (passing actual value as argument)

2. Passing by reference (passing address of an argument)

# Passing structure by value

A structure variable can be passed to the function as an argument as normal variable. If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.

Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function.

#include <stdio.h>

struct student

{

char name[50];

int roll;

};

void Display(struct student stu); /\* function prototype should be below to the structure declaration otherwise compiler shows error \*/

int main()

{

struct student s1;

printf("Enter student's name: ");

scanf("%s",&s1.name);

printf("Enter roll number:");

scanf("%d",&s1.roll);

Display(s1); // passing structure variable s1 as argument

return 0;

}

void Display(struct student stu)

{

printf("Output\nName: %s",stu.name);

printf("\nRoll: %d",stu.roll);

}

Output

Enter student's name: Kevin Amla

Enter roll number: 149

Output

Name: Kevin Amla

Roll: 149

# Passing structure by reference

The address location of structure variable is passed to function while passing it by reference. If structure is passed by reference, change made in structure variable in function definition reflects in original structure variable in the calling function.

Write a C program to add two distances(feet-inch system) entered by user. To solve this program, make a structure. Pass two structure variable (containing distance in feet and inch) to add function by reference and display the result in main function without returning it.

#include <stdio.h>

struct distance

{

int feet;

float inch;

};

void Add(struct distance d1,struct distance d2, struct distance \*d3);

int main()

{

struct distance dist1, dist2, dist3;

printf("First distance\n");

printf("Enter feet: ");

scanf("%d",&dist1.feet);

printf("Enter inch: ");

scanf("%f",&dist1.inch);

printf("Second distance\n");

printf("Enter feet: ");

scanf("%d",&dist2.feet);

printf("Enter inch: ");

scanf("%f",&dist2.inch);

Add(dist1, dist2, &dist3);

/\*passing structure variables dist1 and dist2 by value whereas passing structure variable dist3 by reference \*/

printf("\nSum of distances = %d\'-%.1f\"",dist3.feet, dist3.inch);

return 0;

}

void Add(struct distance d1,struct distance d2, struct distance \*d3)

{

/\* Adding distances d1 and d2 and storing it in d3 \*/

d3->feet=d1.feet+d2.feet;

d3->inch=d1.inch+d2.inch;

if (d3->inch>=12)

{

/\* if inch is greater or equal to 12, converting it to feet. \*/

d3->inch-=12; ++d3->feet;

}

}

Output

First distance

Enter feet: 12

Enter inch: 6.8

Second distance

Enter feet: 5

Enter inch: 7.5

Sum of distances = 18'-2.3"

Example

#include<stdio.h>

#include<conio.h>

void display(struct customer);

struct customer

{

int id;

char name[15];

char address[15];

};

void main()

{

struct customer cu={1,"Binod","Bharatpur"};

clrscr();

display(cu);

getch();

}

void display(struct customer c)

{

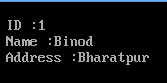
printf("\n ID :%d",c.id);

printf("\n Name :%s",c.name);

printf("\n Address :%s",c.address);

}

Output



**Passing array of structure to function**

We can pass the array of structure to function, where each element of array is of structure type.

Example

#include<stdio.h>

#include<conio.h>

struct customer

{

int id;

char name[15];

char address[15];

};

void display(struct customer c[]);

void main()

{

struct customer cu[3]={{1,"Binod","Bharatpur"},{2,"Nabin","Pokhara"},{3,"Dibya","Kathmandu"}};

clrscr();

display(cu);

getch();

}

void display(struct customer c[])

{

int i;

printf("\nId\tName\tAddress ");

for(i=0;i<3;i++)

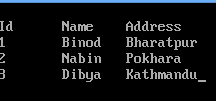
{

printf("\n%d\t%s\t%s",c[i].id,c[i].name,c[i].address);

}

}

Output



* Write a program that reads names and ages of ‘n’ students into the computer and rearrange the names into alphabetical order using the structure variables.

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

struct student

{

char name[15];

int age;

};

struct student st[100],temp;

int i,j,n;

clrscr();

printf("How many students are there: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n Enter student name and age: ");

scanf("%s%d",st[i].name,&st[i].age);

}

for(i=0;i<n-1;i++)

{

for(j=i+1;j<n;j++)

{

if(strcmp(st[i].name,st[j].name)>0)

{

temp=st[i];

st[i]=st[j];

st[j]=temp;

}

}

}

printf("\nName\tAge");

for(i=0;i<n;i++)

{

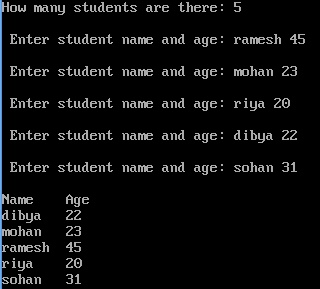
printf("\n%s\t%d",st[i].name,st[i].age);

}

getch();

}

Output



**Union**

Union is a derived data type like structure which can contain members of different data type. Union members share the same memory locations. Compiler allocates sufficient memory to hold the largest member in the union. We can use only one member at a time. Union is used for saving memory. Union can be defined in same manner as structures just the keyword used in defining union in union where keyword used in defining structure was struct.

Syntax for defining union

union union\_name

{

data\_type mem1;

data\_type mem2;

…………………

data\_type memn;

};

Union variables can be created in similar manner as structure variable.

union car{

char name[50];

int price;

}c1, c2, \*c3;

OR;

union car{

char name[50];

int price;

};

-------Inside Function----------

union car c1, c2, \*c3;

In both cases, union variables c1, c2 and union pointer variable c3 of type union car is created. Accessing members of an union

The member of unions can be accessed in similar manner as that structure. Suppose, we you want to access price for union variable c1 in above example, it can be accessed as c1.price. If you want to access price for union pointer variable c3, it can be accessed as (\*c3).price or asc3->price.

Like structure variable union variable is also needed to be declared for accessing members of union.

Syntax

union union\_name union\_variable;

Example

#include<stdio.h>

#include<conio.h>

void main()

{

union student

{

int roll;

char name[15];

};

union student st;

clrscr();

printf("\n Enter roll no: ");

scanf("%d",&st.roll);

printf(" Rollno :%d",st.roll);

printf("\n Enter name: ");

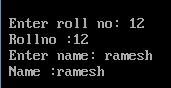
scanf("%s",st.name);

printf(" Name :%s",st.name);

getch();

}

Output



* Difference between structure and union

|  |  |
| --- | --- |
| **Structure** | **Union** |
| 1. Memory occupied by structure is sum of individual data type. | 1. Memory occupied by union is of highest data type of all. |
| 1. Can take part in complex data structure. | 1. Cannot take part in complex data structure. |
| 1. Keyword struct is used. | 1. Keyword union is used. |
| 1. Every element value’s are independent to each other. | 1. If any of the values of any element has been changed there is direct impact to the other elements values. |
| 1. Memory allocation of every element is independent to each other thereby the memory allocation is sum of every element. | 1. Memory allocation is performed by sharing the memory with highest data type. |
| 1. All members can be accessed simultaneously. | 1. Only one member is active at a time, so only one member can be accessed at a time. |
| 1. Syntax   struct struct\_name  {  data\_type mem1;  data\_type mem2;  …………………  data\_type memn;  }; | 1. Syntax   union union\_name  {  data\_type mem1;  data\_type mem2;  …………………  data\_type memn;  }; |

Though unions are similar to structure in so many ways, the difference between them is crucial to understand. This can be demonstrated by this example:

#include <stdio.h>

union job

{ //defining a union

char name[32];

float salary;

int worker\_no;

}u;

struct job1

{

char name[32];

float salary;

int worker\_no;

}s;

int main()

{

printf("size of union = %d",sizeof(u));

printf("\nsize of structure = %d", sizeof(s));

return 0;

}

Output

size of union = 32

size of structure = 40

There is difference in memory allocation between union and structure as suggested in above example. The amount of memory required to store a structure variables is the sum of memory size of all members.

But, the memory required to store a union variable is the memory required for largest element of an union.

What difference does it make between structure and union?

As you know, all members of structure can be accessed at any time. But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.

#include <stdio.h>

union job {

char name[32];

float salary;

int worker\_no;

}u;

int main(){

printf("Enter name:\n");

scanf("%s",&u.name);

printf("Enter salary: \n");

scanf("%f",&u.salary);

printf("Displaying\nName :%s\n",u.name);

printf("Salary: %.1f",u.salary);

return 0;

}

Output

Enter name

Hillary

Enter salary

1234.23

Displaying

Name: f%Bary

Salary: 1234.2

Note: You may get different garbage value of name.

* Difference between array and structure

|  |  |
| --- | --- |
| **Array** | **Structure** |
| 1. Array is a built-in data type. | 1. Structure is a derived data type. |
| 1. Array holds the group of same elements under a single name. | 1. Structure holds the group of different elements under a single name. |
| 1. We cannot have array of array. | 1. We can have array of structure. |
| 1. Memory occupied by an array is the multiple of no of index | 1. Memory occupied by structure is sum of individual data type. |
| 1. Cannot take part in complex data structure. | 1. Can take part in complex data structure. |
| 1. Cannot be used in program to interact with hardware. | 1. Can be used in program to interact with hardware. |
| 1. Syntax for declaration   data\_type arrayname[subscript]; | 1. Syntax for declaration   struct struct\_name  {  data\_type mem1;  data\_type mem2;  …………………  data\_type memn;  }; |

# Pointer to structure

We have already learned that a pointer is a variable which points to the address of another variable of any data type like int, char, float etc. Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable. Here is how we can declare a pointer to a structure variable.

struct dog

{

char name[10];

char breed[10];

int age;

char color[10];

};

struct dog spike;

// declaring a pointer to a structure of type struct dog

struct dog \*ptr\_dog

This declares a pointer ptr\_dog that can store the address of the variable of type struct dog. We can now assign the address of variable spike to ptr\_dog using & operator.

ptr\_dog = &spike;

Now ptr\_dog points to the structure variable spike.