**Structures**

**Structures**: Declaring Structures and Structure variables, Accessing Members of a Structure, Arrays of Structures, Arrays within a Structure

-----------------------------------------------------------------------------------------------------------------

**Why Use Structures**

Theordinary variables can hold one piece of information and how arrays can hold a number of pieces of information of the same data type. These two data types can handle a great variety of situations. But quite often we deal with entities that are collection of dissimilar data types.

**Definition:** A structure is a collection of heterogeneous data items that are stored in consecutive memory locations.

The structure comes under the category of user defined data types.

For example, suppose you want to store data about a book. You might want to store its name (a string), its price (a float) and number of pages in it (an int). If data about say 3 such book is to be stored, then we can follow two approaches:

1. Construct individual arrays, one for storing names, another for storing prices and still

another for storing number of pages.

1. Use a structure variable.

Let us examine these two approaches one by one.. Let us begin with a program that uses *Arrays*.

#include<stdio.h>

void main( )

{

char name[3] ;

float price[3] ;

int pages[3], i ;

printf ( "\nEnter names, prices and no. of pages of 3 books\n" ) ;

for ( i = 0 ; i <= 2 ; i++ )

scanf ( "%c %f %d", &name[i], &price[i], &pages[i] );

printf ( "\nAnd this is what you entered\n" ) ;

for ( i = 0 ; i <= 2 ; i++ )

printf ( "%c %f %d\n", name[i], price[i], pages[i] );

}

**Output:**

Enter names, prices and no. of pages of 3 book And this is what you entered

A 100.00 354 A 100.000000 354

C 256.50 682 C 256.500000 682

F 233.70 512 F 233.700000 512

This approach no doubt allows you to store names, prices and number of pages. But as you must have realized, it is an unwieldy approach that obscures the fact that you are dealing with a group of characteristics related to a single entity—the book.

The program becomes more difficult to handle as the number of items relating to the book go on increasing. For example, we would be required to use a number of arrays, if we also decide to store name of the publisher, date of purchase of book, etc. To solve this problem, C provides a special data type—the structure.

**Defining Structure**

Syntax of structure declaration:

struct structure name

{

*datatype1 var1,var2,…*;

*datatype2 var1,var2,…*;

……………

*datatypen var1,var2,…*;

};

* The struct keyword is used to declare structures.
* The keyword struct identifies the beginning of a structure definition.
* It's followed by a structure name or tag that is the name given to the structure.
* Following the tag are the structure members, enclosed in braces.
* A structure can contain any of C's data types, including arrays and other structures.
* Each variable within a structure is called a *member* of the structure.
* The above structure declaration is also known as template.
* The template is terminated with semicolon (;).

Examples:

1.Define a structure by name student which includes student name,roll number and marks.

struct student

{

char name [10];

int rno;

float marks ;

} ;

This statement defines a new data type called **struct student**.

2. Define a structure by name book which includes name of the book, price and number of pages.

struct book

{

char name ;

float price ;

int pages ;} ;

This statement defines a new data type called **struct book**.

**Declaring Structure Variable**

After defining the structure, variables for that structure are to be declared to access the members of structures.

Syntax for declaration of structure variable is

**struct** structurename **var1,var2,var3,……varn;**

**Example:** struct book b1, b2, b3 ;

The variables **b1**, **b2**, **b3** are variables of the type **struct book.**

The memory is allocated for structures only after the declaration of structure variable. The memory allocated is equal to the sum of memory required by all the members in structure definition.

Example :

struct book b1, b2, b3 ;

This statement assigns space in memory for b1,b2,b3. It allocates space to hold all the elements in the structure—in this case, 7 bytes—one for **name**, four for **price** and two for **pages**. These bytes are always in adjacent memory locations.

Example:

struct student

{

char name [10];

int rno;

float marks ;

} ;

struct student s1,s2;

The above declaration allocates 16 bytes(10 bytes for name,2 bytes for roll number and 4 bytes for marks) for s1 and 16 bytes for s2.

The variables for structure can be declared in three different ways:

**method 1:**

struct structure name

{

*datatype1 var1,var2,…*;

*datatype2 var1,var2,…*;

……………

……………

*datatypen var1,var2,…; };*

**struct** structurename **var1,var2,var3,……varn;**

**method 2:**

struct structure name

{

datatype1 var1,var2,…;

datatype2 var1,var2,…;

…………… .

……………

datatypen var1,var2,…;

}var1,var2,var3,…..varn;

**method3:**

struct

{

datatype1 var1,var2,…;

datatype2 var1,var2,…;

…………… .

……………

datatypen var1,var2,…;

}var1,var2,var3,…..varn;

The method 3 cannot be used for future declaration of variables as structure tag is not present.

Note: The use of structure tag is optional in C language.

For example,Variables for structure book are declared in 3 different ways

1. struct book

{

char name ;

float price ;

int pages ;

} ;

struct book b1, b2, b3 ;

2.struct book

{

char name ;

float price ;

int pages ;

} b1, b2, b3 ;

3. struct

{

char name ;

float price ;

int pages ;

} b1, b2, b3 ;

**Accessing Members of Structure**

The members of structure are accessed by dot (.) operator.

Syntax for accessing members of structure are:

Structure varname.member name;

So to refer to **pages** of the structure defined in our structure book example we have to use,

b1.pages

Similarly, to refer to **price** we would use,

b1.price.

**Structure Initialization**

Like primary variables and arrays, structure variables can also be initialized where they are declared.

Syntax for structure variable initialization:

**struct** structurename **varname={list of values);**

here the values should be assigned in the same order of declaration of the members in the definition.

example:

struct book

{

char name[10] ;

float price;

int pages;

};

struct book b1 = { "Basic", 130.00, 550 } ;

struct book b2 = { "Physics", 150.80, 800 } ;

Examples:

1. Define a structure by name student which includes student name,roll number and marks.

struct student

{

char name[10];

int rno;

float marks ;

} ;

This statement defines a new data type called **struct student**.

2. Define a structure by name book which includes name of the book ,price and number of pages.

struct book

{

char name;

float price;

int pages ;

} ;

This statement defines a new data type called **struct book**.

**Memory allocation for structure**

Whatever be the elements of a structure, they are always stored in contiguous memory locations. The following program would illustrate this:

/\* Memory map of structure elements \*/

main( )

{

struct book

{

char name ;

float price ;

int pages ;

} ;

struct book b1 = { 'B', 130.00, 550 } ;

printf ( "\nAddress of name = %u", &b1.name ) ;

printf ( "\nAddress of price = %u", &b1.price ) ;

printf ( "\nAddress of pages = %u", &b1.pages ) ;

}

Here is the output of the program...

Address of name = 65518

Address of price = 65519

Address of pages = 65523

Actually the structure elements are stored in memory as shown in the Figure below

b1.name b1.price b1.pages

|  |  |  |
| --- | --- | --- |
| B | 130.00 | 550 |

65518 65519 65523

**Difference between Arrays and Structures :**

**Array:**

* Array is a derived data type.
* It allocates memory only for the elements of the subscripts. Array allocates static memory and uses index / subscript for accessing elements of the array.
* It allocates memory of same files that is if we declare integer type array then it allocates 2 byte memory for each cell.
* It does not contain structure within itself.
* It can contain only homogeneous data types.
* Array is a pointer to the first element of it
* We can access the array by using index number.
* The elements of the array are contiguous in memory
* Array element access takes less time in comparison with structures.

**Structure:**

* Structure is a programmer or user - defined data type.
* It does not allocate memory till the elements of the structure are accessed. Structures allocate dynamic memory and uses (.) operator for accessing the member of a structure.
* It allocates the memory for the highest data type.
* It contain array within itself.
* It can contain only non-homogeneous data types.
* Structure is not a pointer
* The elements of structure are accessed by using dot (.) operator with structure reference name.
* The elements of a structure may not be contiguous.

**Sample Program**

Define a structure for student which include roll number ,name,age and marks .

Write a program to read and display the information of 3 students.

#include<stdio.h>

#include<string.h>

struct student

{

int rno;

char name[10];

int marks,age;

};

void main()

{

//assigning values to structure variable s1 using initialization

struct student s1={2,"Gandhi",89,18};

struct student s2,s3;

//assigning values to s2 using assignment operator.

s2.rno=5;

s2.marks=90;

s2.age=18;

strcpy(s2.name,"Ram ");

//reading values for s3 using standard input function.

printf("\n enter rno,name ,marks,age of student: ");

scanf("%d%s%d%d",&s3.rno,s3.name,&s3.marks,&s3.age);

printf("\n\n");

printf("Details of student 1:\n");

printf("\n roll number: %d",s1.rno);

printf("\n name :%s",s1.name);

printf("\n marks: %d",s1.marks);

printf("\n age: %d",s1.age);

printf("\n\n");

printf("Details of student 2:\n");

printf("\n roll number: %d",s2.rno);

printf("\n name :%s",s2.name);

printf("\n marks: %d",s2.marks);

printf("\n age: %d",s2.age);

printf("\n\n");

printf("Details of student 3:\n");

printf("\n roll number: %d",s3.rno);

printf("\n name :%s",s3.name);

printf("\n marks: %d",s3.marks);

printf("\n age: %d",s3.age);

}

**output :**

enter rno,name,marks,age of student:1

Raju

92

18

Details of student 1:

roll number: 2

name :Gandhi

marks :89

age: 18

Details of student 2:

roll number: 5

name :Ram

marks :90

age: 18

Details of student 3:

roll number: 1

name :Raju

marks :92

age: 18

**Operations that can be performed on structure variable**

The only operation that is allowed on structure variables is assignment operation. Two variables of same structure can be copied similar to ordinary variables.

If P1 and P2 are variables of struct P, then P1 values can be assigned to P2 as

P2=P1;

where values of P1 will be assigned to P2 member by member.

Program illustrating operation on structure variables

#include<stdio.h>

#include<string.h>

struct student

{

int rno;

char name[10];

int marks,age;

};

void main()

{

//assigning values to structure variable s1 using initalization

struct student s1={2,"Gandhi",89,18};

struct student s2;

s2=s1;

printf("\nDetails of student 1:\n");

printf("\n roll number: %d",s1.rno);

printf("\n name :%s",s1.name);

printf("\n marks: %d",s1.marks);

printf("\n age: %d",s1.age);

printf("\n\n");

printf("Details of student 2:\n");

printf("\n roll number: %d",s2.rno);

printf("\n name :%s",s2.name);

printf("\n marks: %d",s2.marks);

printf("\n age: %d",s2.age);

}

**output:**

Details of student 1:

roll number: 2

name :Gandhi

marks :89

age: 18

Details of student 2:

roll number: 2

name :Gandhi

marks :89

age: 18

**Note:** C does not permit any logical operations on structure variables.

**Operation On Individual Members Of Structures**

All operations are valid on individual members of structures.

Program illustrating operations on structure members

#include<stdio.h>

#include<string.h>

struct student

{

int rno;

char name[10];

int marks,age;

};

void main()

{

int m;

//assigning values to structure variable s1 using initalization

struct student s1={2,"Gandhi",89,18};

struct student s2;

s2=s1;

printf("\nDetails of student 1:\n");

printf("\n roll number: %d",s1.rno);

printf("\n name :%s",s1.name);

printf("\n marks: %d",s1.marks);

printf("\n age: %d",s1.age);

printf("\n\n");

printf("Details of student 2:\n");

printf("\n roll number: %d",s2.rno);

printf("\n name :%s",s2.name);

printf("\n marks: %d",s2.marks);

printf("\n age: %d",s2.age);

//comparsion of two student details.

m=((s1.rno==s2.rno)&&(s1.marks==s2.marks))?1:0;

if(m==1)

printf("\n both the details are same");

else

printf("\n both the details are not same");

}

**output:**

Details of student 1:

roll number: 2

name :Gandhi

marks :89

age: 18

Details of student 2:

roll number: 2

name :Gandhi

marks :89

age: 18

both the details are same

**Array Of Structures**

In array of structures, the variable of structure is array .

In our sample program, to store details of 100 students we would be required to use 100 different structure variables from **s1** to **s100**,which is definitely not very convenient. A better approach would be to use an array of structures.

**Syntax for declaring structure array**

struct struct-name

{

datatype var1;

datatype var2;

- - - - - - - - - -

- - - - - - - - - -

datatype varN;

};

struct struct-name obj [ size ];

**Program:**

Define a structure for student which include roll number ,name,age and marks .

Write a program to read and display the information of ‘n’ number of students where n is value supplied by user.

#include<stdio.h>

#include<string.h>

struct student

{

int rno;

char name[10];

int marks,age;

};

void main()

{

struct student s[10];//Declares array of 10 student.

int i,n;

printf("\n enter number of students: ");

scanf("%d",&n);

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

{

//reading values for s3 using standard input function.

printf("\n enter rno,name ,marks,age of student %d: ",i+1);

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

}

printf("\n\n");

printf("Details of students are :\n");

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

{

printf("\n Details of student %d:\n",i+1);

printf("\n roll number: %d",s[i].rno);

printf("\n name :%s",s[i].name);

printf("\n marks: %d",s[i].marks);

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

}

}

**Output:**

enter number of students :3

enter rno,name,marks,age of student 1:2

Gandhi

89

18

enter rno,name,marks,age of student 2:5

Raj

76

18

enter rno,name,marks,age of student 3:6

Ram

86

18

Details of student 1:

roll number: 2

name :Gandhi

marks :89

age: 18

Details of student 2:

roll number: 5

name :Raj

marks :76

age: 18

Details of student 3:

roll number: 6

name :Ram

marks :86

age: 18

In the above program the memory allocated for structure variable is 220 bytes consecutively in which first 22 bytes for 1st student 1(s[0]),next 22 bytes for 2nd student 2(s[1]) and so on last 16 bytes for 10 th student (s[9]).

The following figure shows the memory allocation for array of structures.

22bytes 22 22 22 22 22 22 22 22 22

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9]

Again in each 22 bytes (4 bytes-roll number,10 bytes-name,4 bytes-marks,4 bytes-age).

**Array within Structure :**

As we know, structure is collection of different data type. Like normal data type, It can also store an array as well.

In arrays within structure the member of structure is array.

**Syntax for array within structure**

struct struct-name

{

datatype var1; // normal variable

datatype array [size]; // array variable

- - - - - - - - - -

- - - - - - - - - -

datatype varN;

};

struct struct-name obj;

**Example for array within structure**

struct Student

{

int Roll;

char Name[10];

int Marks[3]; //array of marks

int Total;

float Avg;

};

void 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);

}

**Output :**

Enter Student Roll : 4

Enter Student Name : GANDHI

Enter marks 1 : 80

Enter marks 2 : 90

Enter marks 3 : 75

Roll : 4

Name : GANDHI

Total : 245

Average : 81.000000

The memory allocated for structure variable s is 24 bytes(2-roll,10-name,6-marks,2-total,4-avg).

**Structure within a Structure(Nested Structures)**

* Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.
* The structure variables can be a normal structure variable or a pointer variable to access the data.
* This program explains how to use structure within structure in C using normal variable. “student\_college\_detail’ structure is declared inside “student\_detail” structure in this program. Both structure variables are normal structure variables.
* Please note that members of “student\_college\_detail” structure are accessed by 2 dot(.) operator and members of “student\_detail” structure are accessed by single dot(.) operator.

|  |
| --- |
| **Program for Structure within Structure:**  #include <stdio.h>  #include <string.h>    struct student\_college\_detail  {      int college\_id;      char college\_name[50];  };    struct student\_detail  {      int id;      char name[20];      float percentage;      // structure within structure      struct student\_college\_detail clg\_data;  }stu\_data;    int main()  {      struct student\_detail stu\_data = {1, "Raju", 90.5, 71145,"GITAM University"};      printf(" Id is: %d \n", stu\_data.id);      printf(" Name is: %s \n", stu\_data.name);      printf(" Percentage is: %f \n\n", stu\_data.percentage);        printf(" College Id is: %d \n",                      stu\_data.clg\_data.college\_id);      printf(" College Name is: %s \n",                      stu\_data.clg\_data.college\_name);      return 0;  } |

**OUTPUT:**

Id is: 1  
Name is: Raju  
Percentage is: 90.500000

College Id is: 71145  
College Name is: GITAM University

**Example for structure within structure or nested structure:**

#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;

};

void 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\tEmployeeHouseNo : %s",E.Add.HouseNo);

printf("\n\tEmployee City : %s",E.Add.City);

printf("\n\tEmployeeHouseNo : %s",E.Add.PinCode);

}

Output :

Enter Employee Id : 101

Enter Employee Name : Suresh

Enter Employee Salary : 45000

Enter Employee House No : 4598/D

Enter Employee City : Delhi

Enter Employee Pin Code : 110056

Details of Employees

Employee Id : 101

Employee Name : Suresh

Employee Salary : 45000

Employee House No : 4598/D

Employee City : Delhi

Employee Pin Code : 110056

**Structures and pointers**

Pointer to structure holds the address of the entire structure. It is used to create complex data structures such as linked lists, trees, graphs and so on.

The members of the structure can be accessed using a special operator called as an arrow operator ( -> ).

**Declaration**

Following is the declaration for pointers to structures in C programming −

struct tagname \*ptr;

For example − struct student \*s −

**Accessing**

**T**he members of structures can be accessed using pointers with the help of 🡪 operator or **\*.**

ptr-> membername;

Or

(\*ptr).membername;

For example − s->sno, s->sname, s->marks;

**Example Program**

The following program shows the usage of pointers to structures −

#include<stdio.h>

struct student{

   int sno;

   char sname[30];

   float marks;

};

main ( )

{

   struct student s;

   struct student \*st;

   printf("enter sno, sname, marks:");

   scanf ("%d%s%f", & st->sno, st->sname, &st->marks);

   st = &s;

   printf ("details of the student are");

   printf ("Number = %d\n", st ->sno);

   printf ("name = %s\n",(\*st).name);

   printf ("marks =%f\n", (\*st).marks);

   }

**Output**

Let us run the above program that will produce the following result −

enter sno, sname, marks:1 Ram 98

details of the student are:

Number = 1

name = Ram

marks =98.000000

**Structures and Functions**

Structures can be passed as argument to function in 3 ways.

1.Passing members of structures as arguments to function.

2.Passing structure variable as argument to function

3.Passing structure pointer as arguments to function

1.Passing members of structures as arguments to function.

Here individual members of structures are passed as arguments to a function.

#include<stdio.h>

void display(int,float,char []);

struct student

{

int rno;

float m;

char name[10];

};

int main()

{

struct student s={101,6.5,"abc"};

display(s.rno,s.m,s.name);

return 0;

}

void display(int x, float y,char z[])

{

printf("%d\t%f\t%s",x,y,z);

}

**2nd method: passing structure variable as an argument:**

#include<stdio.h>

struct student

{

int rno;

float m;

char name[10];

};

void display(struct student);

int main()

{

struct student s={101,6.5,"abc"};

display(s);

return 0;

}

void display(struct student x)

{

printf("%d\t%f\t%s",x.rno,x.m,x.name);

}

**3rd Method: passing structure pointer as argument to function**

#include<stdio.h>

struct student

{

int rno;

float m;

char name[10];

};

void display(struct student \*);

int main()

{

struct student s={101,6.5,"abc"};

display(&s);

return 0;

}

void display(struct student \*x)

{

printf("%d\t%f\t%s",(\*x).rno,(\*x).m,(\*x).name);

}

**UNIONS**

A union is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

**Defining a Union:**

To define a union, you must use the union statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows –

union [union tag] {

member definition;

member definition;

...

member definition;

} [one or more union variables];

The union tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named Data having three members i, f, and str –

union Data {

int i;

float f;

char str[20];

} d;

Now, a variable of ‘d’ type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, *Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.*

The following example displays the total memory size occupied by the above union −

**Example Program:**

#include <stdio.h>

#include <string.h>

union Data {

int i;

float f;

char str[20];

};

int main( ) {

union Data d;

printf( "Memory size occupied by data : %d\n", sizeof(d));

return 0;

}

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

Memory size occupied by d: 20

**Example program for C union**:

#include <stdio.h>

#include <string.h>

union student

{

            char name[20];

            char subject[20];

            float percentage;

};

int main()

{

    union student record1;

    union student record2;

    // assigning values to record1 union variable

       strcpy(record1.name, "Raju");

       strcpy(record1.subject, "Maths");

       record1.percentage = 86.50;

       printf("Union record1 values example\n");

       printf(" Name       : %s \n", record1.name);

       printf(" Subject    : %s \n", record1.subject);

       printf(" Percentage : %f \n\n", record1.percentage);

    // assigning values to record2 union variable

       printf("Union record2 values example\n");

       strcpy(record2.name, "Mani");

       printf(" Name       : %s \n", record2.name);

       strcpy(record2.subject, "Physics");

       printf(" Subject    : %s \n", record2.subject);

       record2.percentage = 99.50;

       printf(" Percentage : %f \n", record2.percentage);

       return 0;

}

**OUTPUT:**

Union record1 values example

Name : ---------// Garbage Value will be printed

Subject :--------// Garbage Value will be printed

Percentage : 86.500000;

Union record2 values example

Name : Mani

Subject : Physics

Percentage : 99.500000

**Explanation For Above C Union Program**:

There are 2 union variables declared in this program to understand the difference in accessing values of union members.

*Record1 union variable*:

* “Raju” is assigned to union member “record1.name” . The memory location name is “record1.name” and the value stored in this location is “Raju”.
* Then, “Maths” is assigned to union member “record1.subject”. Now, memory location name is changed to “record1.subject” with the value “Maths” (Union can hold only one member at a time).
* Then, “86.50” is assigned to union member “record1.percentage”. Now, memory location name is changed to “record1.percentage” with value “86.50”.
* Like this, name and value of union member is replaced every time on the common storage space.
* So, we can always access only one union member for which value is assigned at last. We can’t access other member values.
* So, only “record1.percentage” value is displayed in output. “record1.name” and “record1.percentage” are empty.

*Record2 union variable:*

* If we want to access all member values using union, we have to access the member before assigning values to other members as shown in record2 union variable in this program.
* Each union members are accessed in record2 example immediately after assigning values to them.
* If we don’t access them before assigning values to other member, member name and value will be over written by other member as all members are using same memory.

We can’t access all members in union at same time but structure can do that.

**Program for one member is being used at a time in 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;

}

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

**Output:**

data.i : 10

data.f : 220.500000

data.str : C Programming

Here, all the members are getting printed very well because one member is being used at a time.

**Difference between union and structure**

Though unions are similar to structure in so many ways, the difference between them is crucial to understand.

|  |  |
| --- | --- |
| **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 greater than or equal to the sum of sizes of its members. The smaller members may end with unused slack bytes. | 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. The address of each member will be in ascending order This indicates that memory for each member will start at different offset values. | 4. The address is same for all the members of a union. This indicates that every member begins at the same offset value. |
| 5 Altering the value of a member will not affect other members of the structure. | 5. Altering the value of any of the member will alter other member values. |
| 6. Individual Structure member can be accessed at a time | 6. Only one Union member can be accessed at a time. |
| 7. Several members of a structure can initialize at once. | 7. Only one member of a union can be initialized. |

The primary difference can be demonstrated by this example:

#include <stdio.h>

union unionJob

{

//defining a union

char name[32];

float salary;

int workerNo;

} uJob;

struct structJob

{

char name[32];

float salary;

int workerNo;

} sJob;

int main()

{

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

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

return 0;

}

**Output**

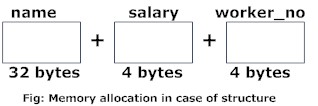
size of union = 32

size of structure = 40

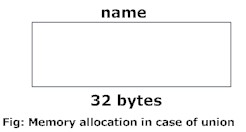
More memory is allocated to structures than union

As seen in the above example, there is a difference in memory allocation between union and structure.

The amount of memory required to store a structure variable is the sum of memory size of all members.



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



**Sample Program with Structures & Union:**

#include<stdio.h>

struct number1

{

int i;

float f;

char c;

}sn;

union number2

{

int i;

float f;

char c;

}un;

void main()

{

printf("Details of structure\n");

printf("size of structure number1=%d\n",sizeof(sn));

sn.i=10;

sn.f=89.00;

sn.c='X';

printf("sn.i=%d\n sn.f=%f\n sn.c=%c\n",sn.i,sn.f,sn.c);

printf("Details of Union\n");

printf("size of Union number2=%d\n",sizeof(un));

un.i=10;

un.f=89.00;

un.c='X';

printf("un.i=%d\n un.f=%f\n un.c=%c\n",un.i,un.f,un.c);

}

**Output:**

size of structure number1= 7

sn.i =10

sn.f =89.00

sn.c =X

size of Union number2 = 4

un.i =Garbage value

un.f = Garbage value

un.c =X