UNIONS

Definition -

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.*

Syntax for Defining a Union:

- ✓ The union statement defines a new data type with more than one member for your program.

```
union union_name
{
      member definition;
      member definition;
      ...
      member definition;
} [one or more union variables];
```

- ✓ The union name (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.

Example:

```
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.*

Example Program: To displays the total memory size occupied by the union.

```
#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;
 }
```

Output:

Memory size occupied by d: 20

Example Program: Accessing Union Members.

```
#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:

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.

Example Program: To access one union member at a time.

```
#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
```

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

Fig: Memory allocation in case of structure

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

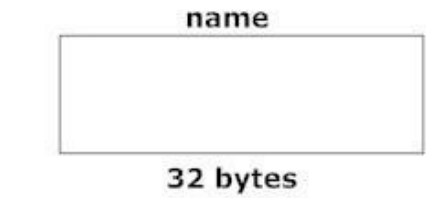

Fig: Memory allocation in case of union

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
```