

## 5.5 UNIONS IN C

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.


All the members of a union share the same memory location. Therefore, if we need to use the same memory location for two or more members, then union is the best data type for that. The largest union member defines the size of the union.

### a) Defining a Union

Union variables are created in same manner as structure variables. The keyword **union** is used to define unions in C language.

#### **Syntax**

Here is the syntax to define a **union** in C language –



```
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 "inti;" 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.

### b) Accessing the Union Members

To access any member of a union, we use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use the keyword union to define variables of union type.

### Syntax

Here is the syntax to access the members of a union in C language –

```
union_name.member name;
```

### c) Initialization of Union Members

You can initialize the members of the union by assigning the value to them using the assignment (=) operator.

### Syntax

Here is the syntax to initialize members of union –

```
union variable.member name= value;
```

### Example

The following code statement shows to initialization of the member "i" of union "data"

```
-data.i=10;
```

### a) Examples of Union

### Example 1

The following example shows how to use unions in a program –



```
#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");

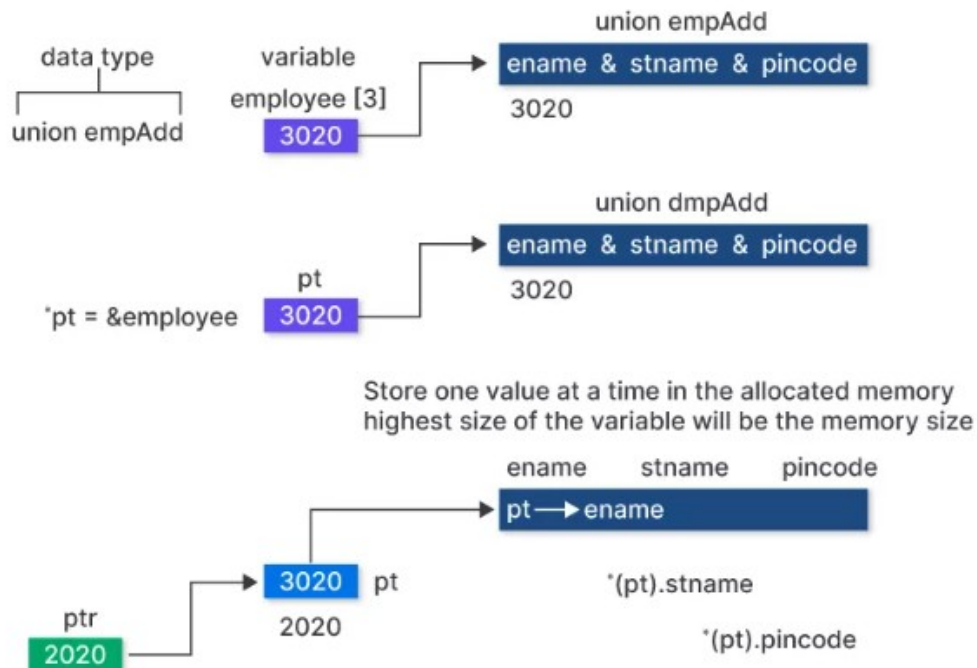
    printf("data.i: %d \n", data.i);
    printf("data.f: %f \n", data.f);
    printf("data.str: %s \n", data.str);
    return 0;
}
```

### *Output*

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

```
data.i: 1917853763
data.f: 4122360580327794860452759994368.000000
data.str: C Programming
```

Here, we can see that the values of **i** and **f** (members of the union) show **garbage values** 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.

Accessing union Members in C

When you define a union, you're essentially creating a container for multiple members, each of which can have a different data type. These members share the same memory space that you can use to store any of the union's members interchangeably. But you can actively access only one of the members. There are two methods of accessing members of a union in C-

1. Using the dot operator (.)
2. Using the arrow operator/ 'this' pointer (->)

However, note that accessing members of a union in C programs involves working with a specific/ distinct type of data. The most commonly used member types are:

1. Integer Members: These members store whole numbers (integer type).
2. Floating-Point Members: These members store decimal numbers (floats or doubles).
3. Character Members: These members store individual characters.
4. Array Members: These members store collections of elements of the same data type.

5. **Pointer Members:** These members store memory addresses, pointing to specific locations in memory.

We can use the dot operator to access most normal data types of union member's values, including integer, double, character, string (character array) and floating-point values. But when working with pointer type members/ union variables, we must use the arrow pointer method. Let's look at both methods!

### **Accessing Normal Members of Union in C Using Dot Operator**

Accessing normal members of a union in C using the dot operator (.) is straightforward and is used when the union variable itself is not a pointer. All you have to do is use the dot operator (.) followed by the member name to access and manipulate the members of the union.

#### **Syntax:**

VariableName.member\_name;



Here, we specify the name of the union variable (variableName) and the name of the union member (member\_name), connected with a dot operator. We have already seen how to use this approach in the example above, but here is another sample C program demonstrating how to access members of a union in C using the dot operator: