Unions

In this tutorial, you will master all about the union in C which is quite similar to the structure. You will cover the topics like how to define and access union, how the union differs from structure etc with the support of super easy examples.

Unions in C

Union is a user-defined data type in C, just like struct, which gathers single or different data types. Like struct, the union also has a specific name with data elements as its members.

Syntax:

The prototype of the definition is

```
union union_name
{
    data_type member_1;
    data_type member_2;
    ...
    ...
    data_type member_n;
};
```

where the `union` is a keyword that informs the compiler that a union is defined in the program. `union_name` stands for the specific name of the union which is used to declare the elementary variable of this particular type. `member_1` , `member_2` ,.., `member_n` are union members of the same or different data types. Union members must be declared inside curly braces `{}` and should terminate by a semicolon`(;)`. Last but not least the union definition must end with a semicolon soon after the closing brace.

Here is the example of union definition:

```
union employee
{
  int id_no;
    char name[20];
    float salary;
};
```

This is how we define a union. Here the `employee` is a union containing 3 different members namely, `id_no` of integer data type, `name` of character data type, and `salary` of float data type.

How to declare a union

Remember the fact that defining alone does not reserve any memory for the union. To make the union active and work with it we need to declare its variables. The variable declaration can be done in two ways :

1. **During union definition**

```
union union_name
{
    data_type member_1;
    data_type member_2;
    ...
    ...
    data_type member_n;
}var_1, var_2;
```

Here is the example

```
union employee
{
  int id_no;
    char name[20];
    float salary;
}emp_1,emp_2;
```

2. **Using the keyword union inside the main() function**

```
union union_name var_1,var_2;
```

Here is the example

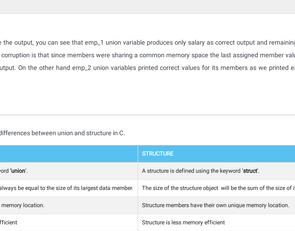
```
int main()
{
union employee emp_1,emp_2;
 }
```

In both instances, we have declared two variables emp_1 and emp_2.

Memory allocation of Union Variable

The unions look quite similar to the structure, but the key difference is that more than one member variable of a union cannot be used at the same time. It is because all the member variables of a union share the same space. More specifically, when a union variable is declared, the compiler will allocate space for the variable consuming the largest space. The rest of the variables will share that space whenever required.

Here is a simple demonstration



When `emp_1` is declared, the compiler will allocate the space for the variable `name[20]` as it acquires more space i.e. 20 bytes of memory. The remaining `id_no` and `salary` will share the required 4 bytes of space from these 20 bytes.

How to access union members

Just like structure, member variables of the union are accessed by member access operator or dot operator(.) and the pointer variables of the union are accessed using arrow pointer(->). The name of the member variables is written as subscript by mentioning it after the union variable name followed by a period.

```
var_1.member_1;
var_1.member_2;
```

Key point to be stored in mind while accessing member variables is that we can only access one member at a time. Lets see what will happen if we attempt to access multiple variables at a time.

```
#include <stdio.h>
#include <string.h>

union Employee
{
            char name[20];
            int id;
            float salary;
};

int main()
{
    union Employee emp_1,emp_2;


    // assigning values to emp_1 union variable at a time
      strcpy(emp_1.name, "TOM");
      emp_1.id = 1001;
      emp_1.salary = 8000;

      printf("Employee 1 Details \n");
      printf("----------------\n");
      printf(" Name      : %s \n", emp_1.name);
      printf(" ID        : %d\n", emp_1.id);
      printf(" Salary : %f \n\n", emp_1.salary);

    // assigning values to emp_2 union variables one after other
      printf("Employee 2 Details\n");
      printf("----------------\n");
      strcpy(emp_2.name, "Jerry");
      printf(" Name      : %s \n", emp_2.name);

      emp_2.id = 1002;
      printf(" ID        : %d\n", emp_2.id);

      emp_2.salary = 9000;
      printf(" Salary : %f \n", emp_2.salary);
      return 0;
}
```

The output will be:

```
Employee 1 Details
----------------
 Name      : 1174011904
 ID        : 8000.000000
 Salary

Employee 2 Details
----------------
 Name      : Jerry
 ID        : 1002
 Salary : 9000.000000
```

Explanation: When you observe the output, you can see that emp_1 union variable produces only salary as correct output and remaining members produces corrupted output. The reason behind this corruption is that since members were sharing a common memory space the last assigned member value will be stored in the memory and hence produce the right output. On the other hand emp_2 union variables printed correct values for its members as we printed each member immediately after assigning them.

Union Vs Structure

Following the table list the key differences between union and structure in C.

UNION	STRUCTURE
A union is defined using the keyword '**union**'.	A structure is defined using the keyword '**struct**'.
The size of the union object will always be equal to the size of its largest data member.	The size of the structure object will be the sum of the size of its data members.
Union members share a common memory location.	Structure members have their own unique memory location.
Union is more memory memory efficient	Structure is less memory efficient
Value initialized to last member will be stored in shared memory	Value initialized to all members will be stored in their distinct memory.
At a time , only one member can be accessed.	At a time, all members can be accessed.
Syntax: `union union_name { data_type member_1; data_type member_2; ... ... data_type member_n; };`	**Syntax:** `struct struct_name { data_type member_1; data_type member_2; ... ... data_type member_n; };`
Example: `union Employee { char name[20]; // 20 bytes int id;  // 4 bytes float salary;  // 4 bytes }; //union object size is 20 bytes`	**Example:** `struct Employee { char name[20]; // 20 bytes int id;  // 4 bytes float salary;  // 4 bytes }; // structure object size is 28 bytes.`