C Unions

A union is a user-defined type similar to structs in C except for one key difference. Structures allocate enough space to store all their members, whereas **unions can only hold one member value at a time**.

How to define a union?

We use the `union` keyword to define unions. Here's an example:

```
union car
{
  char name[50];
  int price;
};
```

The above code defines a derived type `union car`.

Create union variables

When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Here's how we create union variables.

```
union car
{
  char name[50];
  int price;
};
```

```
int main()
{
  union car car1, car2, *car3;
  return 0;
}
```

Another way of creating union variables is:

```
union car
{
  char name[50];
  int price;
} car1, car2, *car3;
```

In both cases, union variables `car1`, `car2`, and a union pointer `car3` of `union car` type are created.

Access members of a union

We use the `.` operator to access members of a union. And to access pointer variables, we use the `->` operator.
In the above example,

- To access `price` for `car1`, `car1.price` is used.
- To access `price` using `car3`, either `(*car3).price` or `car3->price` can be used.

Difference between unions and structures

Let's take an example to demonstrate the difference between unions and structures:

```c
#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 bytes", sizeof(uJob));
   printf("\nsize of structure = %d bytes", sizeof(sJob));
   return 0;
}
```

Output

```
size of union = 32
size of structure = 40
```

Why this difference in the size of union and structure variables?

Here, the size of `sJob` is 40 bytes because

- the size of `name[32]` is 32 bytes
- the size of `salary` is 4 bytes
- the size of `workerNo` is 4 bytes

However, the size of `uJob` is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (`name[32]`), is 32 bytes.

With a union, all members share **the same memory**.

Example: Accessing Union Members

```c
#include <stdio.h>
union Job {
   float salary;
   int workerNo;
} j;

int main() {
   j.salary = 12.3;

   // when j.workerNo is assigned a value,
   // j.salary will no longer hold 12.3
   j.workerNo = 100;

   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

Output

```
Salary = 0.0
Number of workers = 100
```
