Problem Solving With C

UE15CS151

UNIONS

This Notes can be used only for reference and kindly do not solely depend on it. Only those topics which need more explanation are included here. Please Note "The prescribed Text book has to be referred for the examination"

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  tag {
  member definition;
  member definition;
  ...
  member definition;
} one or more union variables;
```

```
union Data {

  int i;

  float f;

   double d;

} d1;
```

Now, a variable of **Data** type can store an integer, a floating-point number, or a double value. 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 8 bytes of memory space because this is the maximum space which can be occupied by a double type. The following example displays the total memory size occupied by the above union −

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

union Data {
  int i;
  float f;
  double d;
};

int main( ) {

  union Data d1;

  printf( "Memory size occupied by data : %d\n", sizeof(1));

  return 0;
}
```

Output : **Memory size occupied by data : 8**

Accessing 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. The following example shows how to use unions in a program −

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

union Data {
  int i;
  float f;
  double d;
};

int main( ) {

  union Data data;

  data.i = 10;
  data.f = 220.5;
```

```
   data.d = 123.456;
   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.d : %lf\n", data.d);

   return 0;
}
```

Output :

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.d : 123.456
```

Here, we can see that the values of **i** and **f** members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of **d** member is getting printed very well.

Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having unions −

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

union Data {
   int i;
   float f;
   double d
};

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);
```

```
  double.d  = 1234.456;
  printf( "data.d : %lf\n", data.d);

  return 0;
}
```

output :

```
data.i : 10
data.f : 220.500000
data.d = 1234.456
```

Union with in a structure :

```
#include<stdio.h>
#include<stdlib.h>
union B
{
        char a[4];
        double d;
}b1;

struct  A
{
        int a;
        float b;
        union B b1;
}a1;
int main()
{
        printf("Size of the structure is %d\n",sizeof(a1));
        a1.a = 10;
        a1.b = 67.67;
        strcpy(a1.b1.a, "XYZ");
        a1.b1.d= 56.3;
        printf("%d %f %s %lf\n", a1.a, a1.b, a1.b1.a, a1.b1.d);

}
```

Unions can have pointers, we can create array of unions , nested union etc , the syntax is similar to that of the structure , only the usage with respect to the memory will vary