C Programming – C Union – Syntax and Examples

C Union

- Union is an user defined datatype in C programming language.
- It is a collection of variables of different datatypes in the same memory location.
- We can define a union with many members, but at a given point of time only one member can contain a value.
- Unions can be very handy when you need to talk to peripherals through some memory mapped registers.

Need for Union in C programming

C unions are used to save memory. To better understand an union, think of it as a chunk of memory that is used to store variables of different types. When we want to assign a new value to a field, then the existing data is replaced with new data.

C unions allow data members which are mutually exclusive to share the same memory. This is quite important when memory is valuable, such as in embedded systems. Unions are mostly used in embedded programming where direct access to the memory is needed.

Difference between structure and union

The main difference between structure and a union is that

- Structs allocate enough space to store all of the fields in the struct. The first one is stored at the beginning of the struct, the second is stored after that, and so on.
- Unions only allocate enough space to store the largest field listed, and all fields are stored at the same space .

Syntax for Declaring a C union

Syntax for declaring a union is same as that of declaring a structure except the keyword struct.

```
union union_name {
    datatype field_name;
    datatype field_name;
    // more variables
};
```

Note : Size of the union is the the size of its largest field because sufficient number of bytes must be reserved to store the largest sized field.

To access the fields of a union, use dot(.) operator i.e., the variable name followed by dot operator followed by field name.

Initializing unions:

The difference between structure and union when their initialization is given below.

```
cUnionExample.c
#include <stdio.h>

struct test1 {
    int x, y;
 };

union test {
    int x, y;
 };

int main() {
    struct test1 t1={1,2};
    union test t;

    t.x = 3;       // t.y also gets value 3
    printf ("after fixing x value the coordinates of t will be  %d %d\n\n",t.x, t.y);

    t.y = 4;  // t.x is also updated to 4
    printf ("After fixing y value the coordinates of t will be  %d %d\n\n", t.x, t.y);

    printf("The coordinates of t1 are %d %d",t1.x,t1.y);

    return 0;
 }
```

```
Output
after fixing x value the coordinates of t will be  3 3
After fixing y value the coordinates of t will be  4 4
The coordinates of t1 are 1 2
```

The fields of a union cannot be initialized all at once. For a structure variable the output is fine but for the union variable the answer does not seem to be correct because union can hold single value for a member at a time i.e., the data **is over-written in the memory**. Obviously the use case we have chosen is not suited for union but only struct. This example is only to demonstrate the behavior of union and structure.

Unions inside structure

In the below code, we have union embedded with in a structure. We know, the fields of a union will share memory, so in main program we ask the user which data he/she would like to store and depending on the user choice the appropriate field will be used. By this way we can use the memory efficiently.

```
cUnionExample.c
```

```
#include<stdio.h>

struct student {
    union {            //anonymous union (unnamed union)
        char name[10];
        int roll;
    };

    int mark;
 };

int main() {
    struct student stud;
    char choice;

    printf("\n You can enter your name or roll number ");
    printf("\n Do you want to enter the name (y or n): ");
    scanf("%c",&choice);

    if(choice=='y'||choice=='Y') {
        printf("\n Enter name: ");
        scanf("%s",stud.name);
        printf("\n Name:%s",stud.name);
    }

    else {
        printf("\n Enter roll number");
        scanf("%d",&stud.roll);
        printf("\n Roll:%d",stud.roll);
    }

    printf("\n Enter marks");
    scanf("%d",&stud.mark);

    printf("\n Marks:%d",stud.mark);

    return 0;
 }
```

```
Output
You can enter your name or roll number

Do you want to enter the name (y or n) : y

Enter name: john

Name:john

Enter marks: 45

Marks:45
```

Structures inside Unions

Example for defining a structure inside union is given below :

```
cUnionExample.c
```

```
#include<stdio.h>

int  main() {

    struct student {
        char name[30];
        int rollno;
        float percentage;
    };

    union details {
        struct student s1;
    };

    union details set;

    printf("Enter details:");

    printf("\nEnter name : ");
    scanf("%s", set.s1.name);

    printf("\nEnter roll no : ");
    scanf("%d", &set.s1.rollno);

    printf("\nEnter percentage :");
    scanf("%f", &set.s1.percentage);

    printf("\nThe student details are : \n");
    printf("\name : %s", set.s1.name);
    printf("\nRollno : %d", set.s1.rollno);
    printf("\nPercentage : %f", set.s1.percentage);

    return 0;
 }
```

Conclusion

In this C Tutorial, we have learnt about C unions, their syntax and usage. Also, we learnt how C unions are different from C structures.