Union in C

C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

Union and structure in C are same in concepts, except allocating memory for their members. Structure allocates storage space for all its members separately.
Whereas, Union allocates one common storage space for all its members.

We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.
Many union variables can be created in a program and memory will be allocated for each union variable separately.
Below table will help you how to form a C union, declare a union, initializing and accessing the members of the union.

Using normal variable	**Using pointer variable**

Syntax:
union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
	Syntax:
	union tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};

Example:
union student
{
int mark;
char name[10];
float average;
};
	Example:
	union student
	{
	int mark;
	char name[10];
	float average;
	};

Declaring union using normal variable: union student report;	**Declaring union using pointer variable:** union student *report, rep;
Initializing union using normal variable: union student report = {100, "Meni", 99.5};	**Initializing union using pointer variable:** union student rep = {100, "Meni", 99.5}; report = &rep;
Accessing union members using normal variable: report.mark;	**Accessing union members using pointer variable:** report -> mark;

report.name; report.average;	report -> name; report -> average;

EXAMPLE PROGRAM FOR C UNION:

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

union student
{
        char name[20];
        char subject[20];
        float percentage;
};

int main()
{
   union student record1;
   union student record2;

   // assigning values to record1 union variable
     strcpy(record1.name, "Princy");
     strcpy(record1.subject, "Prog. in C");
     record1.percentage = 96.50;
```

96.50

```
    record1.percentage
     printf("Union record1 values example\n");
     printf(" Name       : %s \n", record1.name);
     printf(" Subject    : %s \n", record1.subject);
     printf(" Percentage : %f \n\n", record1.percentage); // 96.50

   // assigning values to record2 union variable
     printf("Union record2 values example\n");
     strcpy(record2.name, "Meni");
     printf(" Name       : %s \n", record2.name);

     strcpy(record2.subject, "Physics");
     printf(" Subject    : %s \n", record2.subject);

     record2.percentage = 99.50;
     printf(" Percentage : %f \n", record2.percentage);
     return 0;
}
```

OUTPUT:

Union record1 values example

Name :
Subject :
Percentage : 96.500000;
Union record2 values example
Name : Meni
Subject : Physics
Percentage : 99.500000

EXPLANATION FOR ABOVE C UNION PROGRAM:
There are 2 union variables declared in this program to understand the difference in accessing values of union members.

Record1 union variable:
"Raju" is assigned to union member "record1.name" . The memory location name is "record1.name" and the value stored in this location is "Raju".
Then, "Prog in C" is assigned to union member "record1.subject". Now, memory location name is changed to "record1.subject" with the value "Prog. in C" (Union can hold only one member at a time).
Then, "96.50" is assigned to union member "record1.percentage". Now, memory location name is changed to "record1.percentage" with value "96.50".
Like this, name and value of union member is replaced every time on the common storage space. So, we can always access only one union member for which value is assigned at last. We can't access other member values.
So, only "record1.percentage" value is displayed in output. "record1.name" and "record1.percentage" are empty.
Record2 union variable:
If we want to access all member values using union, we have to access the member before assigning values to other members as shown in record2 union variable in this program.
Each union members are accessed in record2 example immediately after assigning values to them.
If we don't access them before assigning values to other member, member name and value will be over written by other member as all members are using same memory.
We can't access all members in union at same time but structure can do that.
EXAMPLE PROGRAM – ANOTHER WAY OF DECLARING C UNION:
In this program, union variable "record" is declared while declaring union itself as shown in the below program.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  union student
5  {
6          char name[20];
7          char subject[20];
8          float percentage;
9  }record;
10
```

```
11 int main()
12 {
13
14          strcpy(record.name, "Puja");
15          strcpy(record.subject, "Maths");
            record.percentage = 86.50;

            printf(" Name       : %s \n", record.name);
            printf(" Subject    : %s \n", record.subject);
            printf(" Percentage : %f \n", record.percentage);
            return 0;
    }
```

OUTPUT:

Name :
Subject :
Percentage : 86.500000

NOTE:

We can access only one member of union at a time. We can't access all member values at the same time in union.

But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.

DIFFERENCE BETWEEN STRUCTURE AND UNION IN C:

C Structure	**C Union**
Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
Structure occupies higher memory space.	Union occupies lower memory space over structure.
We can access all members of structure at a time.	We can access only one member of union at a time.

Structure example:
struct student
{
int mark;
char name[6];
double average;
};
	Union example:
	union student
	{
	int mark;
	char name[6];
	double average;
	};

For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

Functions returning Pointer variables

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function. They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

```
#include <stdio.h>

int* larger(int*, int*);

void main()
{
   int a = 15;
   int b = 92;
   int *p;
   p = larger(&a, &b);
   printf("%d is larger", *p);
}

int* larger(int *x, int *y)
{
   if(*x > *y)
      return x;
   else
      return y;
}
```

92 is larger

Example of Pointer to Function

```
#include <stdio.h>

int sum(int x, int y)
{
   return x+y;
}

int main( )
```

```
{
    int (*fp)(int, int);
    fp = sum;
    int s = fp(10, 15);
    printf("Sum is %d", s);

    return 0;
}
```

25