A **union** is a special data type available in C that allows to store different data types in the same memory location. We 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 major difference between union and structure is in terms of memory allocation is each member of a structure get its separate memory location ,whereas all members in a union share the same memory location of the biggest number.**

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 [union tag] {

member definition;

member definition;

...

member definition;

} [one or more union variables];

The **union tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the union's definition, before the final semicolon, we can specify one or more union variables but it is optional.

Here is the way you would define a union type named Data having three members i, f, and str −

union Data {

int i;

float f;

char str[20];

} data;

Now, a variable of **Data** type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. we 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 **20 bytes** of memory space because this is the **maximum space** which can be occupied by a character string. The following example displays the total memory size occupied by the above union −

[Live Demo](http://tpcg.io/5ndKkD)

#include <stdio.h>

#include <string.h>

union Data {

int i; //size of int 4 bytes.

float f; //size of int 4 bytes

char str[20]; //total size 20 bytes

};

int main( )

{

union Data data;

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

return 0;

}

When the above code is compiled and executed, it produces the following result −

Memory size occupied by data : 20

**ANOTHER EXAMPLE:**

#include <stdio.h>

#include <string.h>

union student {

int roll; // size of int 4 bytes

float height; //size of float 4bytes

char gender; //size of char 1 bytes

};

int main( )

{

union student data;

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

return 0;

}

**Output:**

memory size occupied by data : 4 bytes

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 −

[Live Demo](http://tpcg.io/KteThe)

#include <stdio.h>

#include <string.h>

union Data {

int i;

float f;

char str[20];

};

int main( )

{

union Data data;

data.i = 10;

data.f = 220.5;

strcpy( data.str, "C Programming");

printf( "data.i : %d\n", data.i);

printf( "data.f : %f\n", data.f);

printf( "data.str : %s\n", data.str);

return 0;

}

When the above code is compiled and executed, it produces the following result −

data.i : 1917853763

data.f : 4122360580327794860452759994368.000000

data.str : C Programming

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 **str** 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 −

[Live Demo](http://tpcg.io/1T4bqA)

#include <stdio.h>

#include <string.h>

union Data {

int i;

float f;

char str[20];

};

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);

strcpy( data.str, "C Programming");

printf( "data.str : %s\n", data.str);

return 0;

}

When the above code is compiled and executed, it produces the following result −

data.i : 10

data.f : 220.500000

data.str : C Programming

Here, all the members are getting printed very well because one member is being used at a time.

**Another example**

#include<stdio.h>

union student

{

int roll;

char gender;

float height;

}s2; //union variable s2 declaration

void main()

{

union student s1; //union ,variable s1 declaration

s1.roll=25;

printf("%d",s1.roll);

s1.gender='M';

printf("\n %c",s1.gender);

}

Output:

25

M

/ /**UNION VARIABLE CAN BE INITIALIZED ONLY FOR THE FIRST MEMBER AND REST OF THE MEMBERS CAN BE INITIALIZED BY ASSIGNING VALUES OR BY TAKING INPUT FROM KEYBOARD**

#include<stdio.h>

union student

{

int roll;

char gender;

float height;

};

void main()

{

// union student1 s1={13,'f',5.5}; //error invalid initialization

union student s2={1}; //valid initialization forthe first member roll

printf("roll=%d ",s2.roll);

union student s3;

printf("\n enter gender \n");

scanf("%c",&s3.gender);

printf("\n gender=%c",s3.gender);

}

**Output:**

**Roll=1**

**Enter gender M**

**Gender=M**

//**Acessing the members of a union—to access members of union same syntax can be followed as with structure i.e use of dot operator or period operator along with the union variable as shown below**

#include<stdio.h>

union student

{

int roll;

char gender;

float height;

};

void main()

{

Union student s1; //union variable s1 declaration

s1.roll=34; //ccessiong member roll & initializing

printf("\n roll=%d",s1.roll);

s1.gender='M'; //accessing member gender & initializing

printf("\n gender=%c",s1.gender);

s1.height=5.3; //acessing memeber height & initializing

printf("\n height=%f",s1.height);

}

**Output:**

**Roll=34**

**Gender=M**

**Height =5.3**