

Unions and Bitwise Operator in C

By: Atiya Jokhio

Unions

A union is a derived data type—like a structure, with members that share the same storage space.

For different situations in a program, some variables may not be relevant, but other variables are—so a union shares the space instead of wasting storage on variables that are not being used.

The members of a union can be of any data type. The number of bytes used to store a union must be at least enough to hold the largest member.

In most cases, unions contain two or more data types. Only one member, and thus one data type, can be referenced at a time.

Unions Declaration

A union definition has the same format as a structure definition. The union definition

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

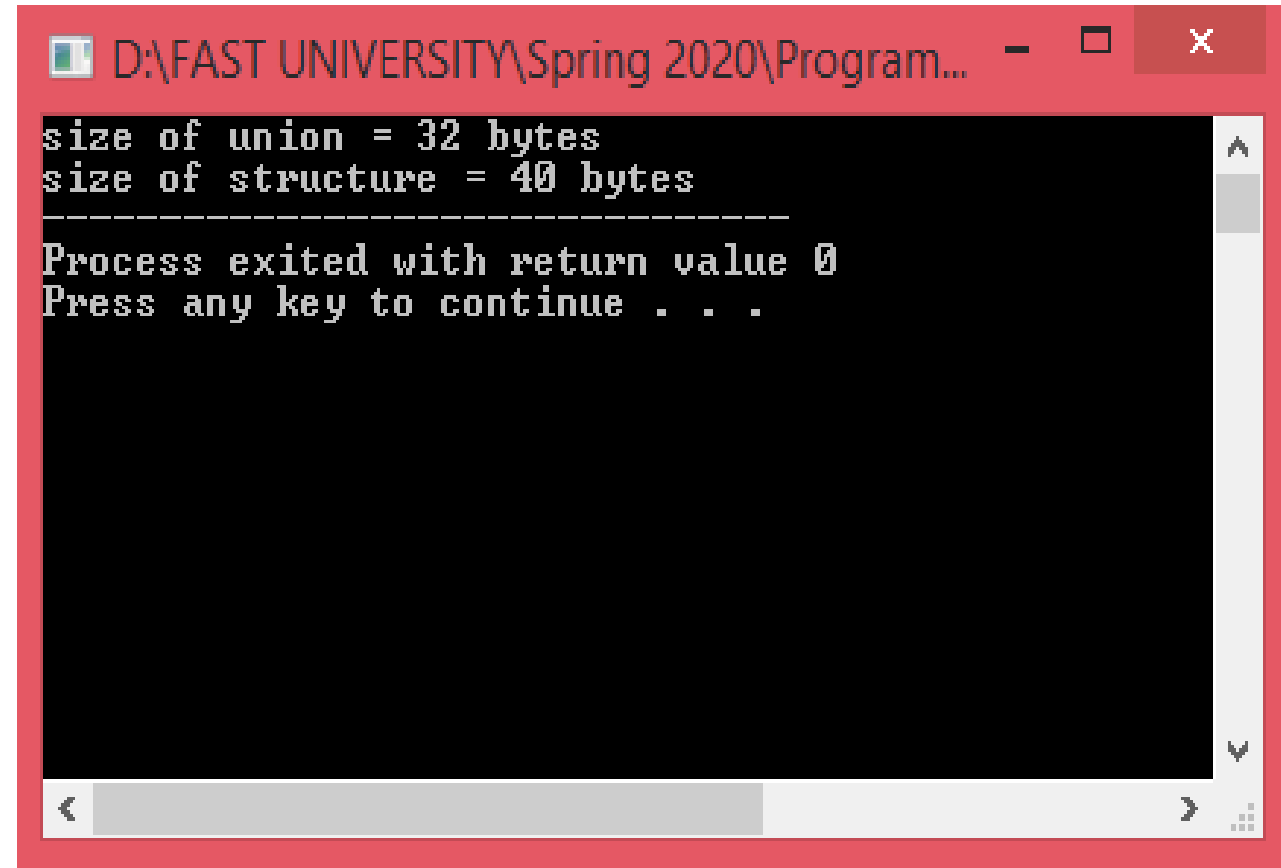
Unions

Memory allocations

```
#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;
}
```



```
D:\FAST UNIVERSITY\Spring 2020\Program...
size of union = 32 bytes
size of structure = 40 bytes
-----
Process exited with return value 0
Press any key to continue . . .
```

Unions

Example

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

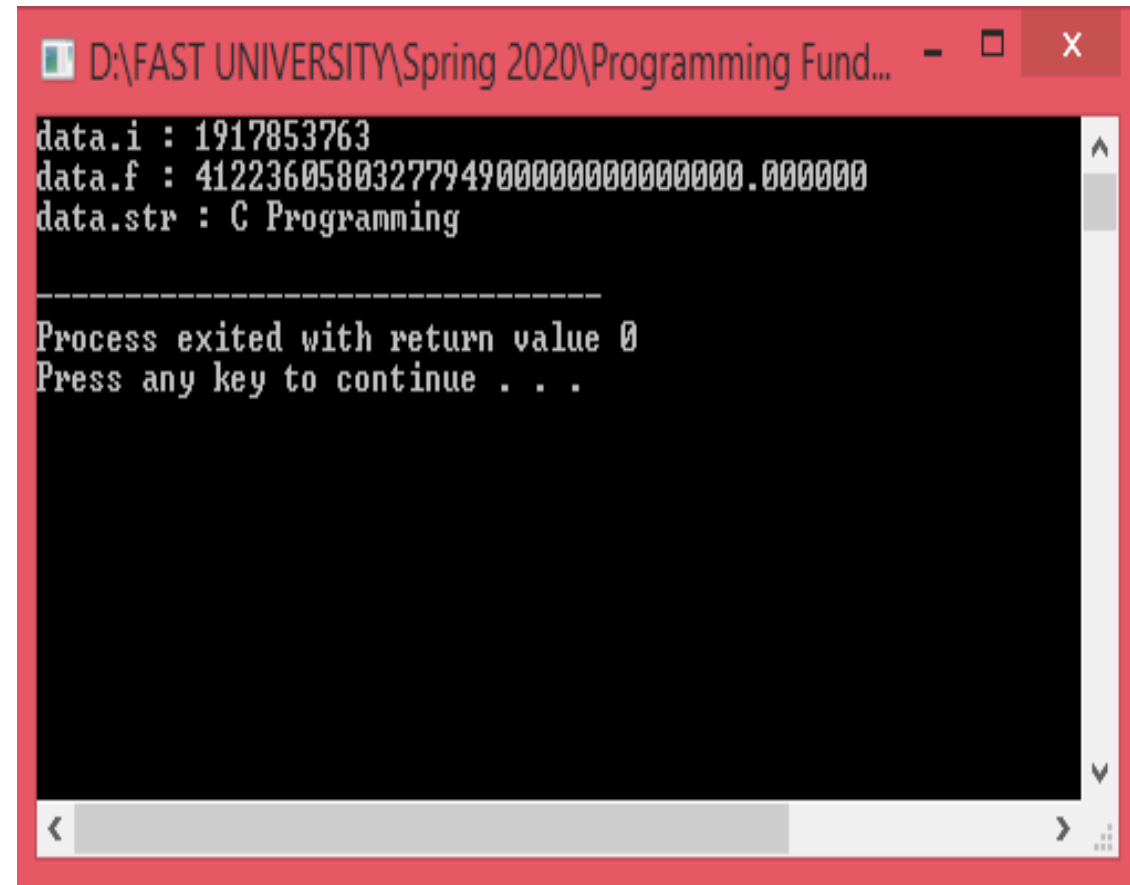
union Data {
    int i;
    float f;
    char str[20];
};

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;
}
```



```
D:\FAST UNIVERSITY\Spring 2020\Programming Fund...
data.i : 1917853763
data.f : 41223605803277949000000000000000.000000
data.str : C Programming
-----
Process exited with return value 0
Press any key to continue . . .
```

Unions Example

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

union Data {
    int i;
    float f;
    char str[20];
};

int main( ) {

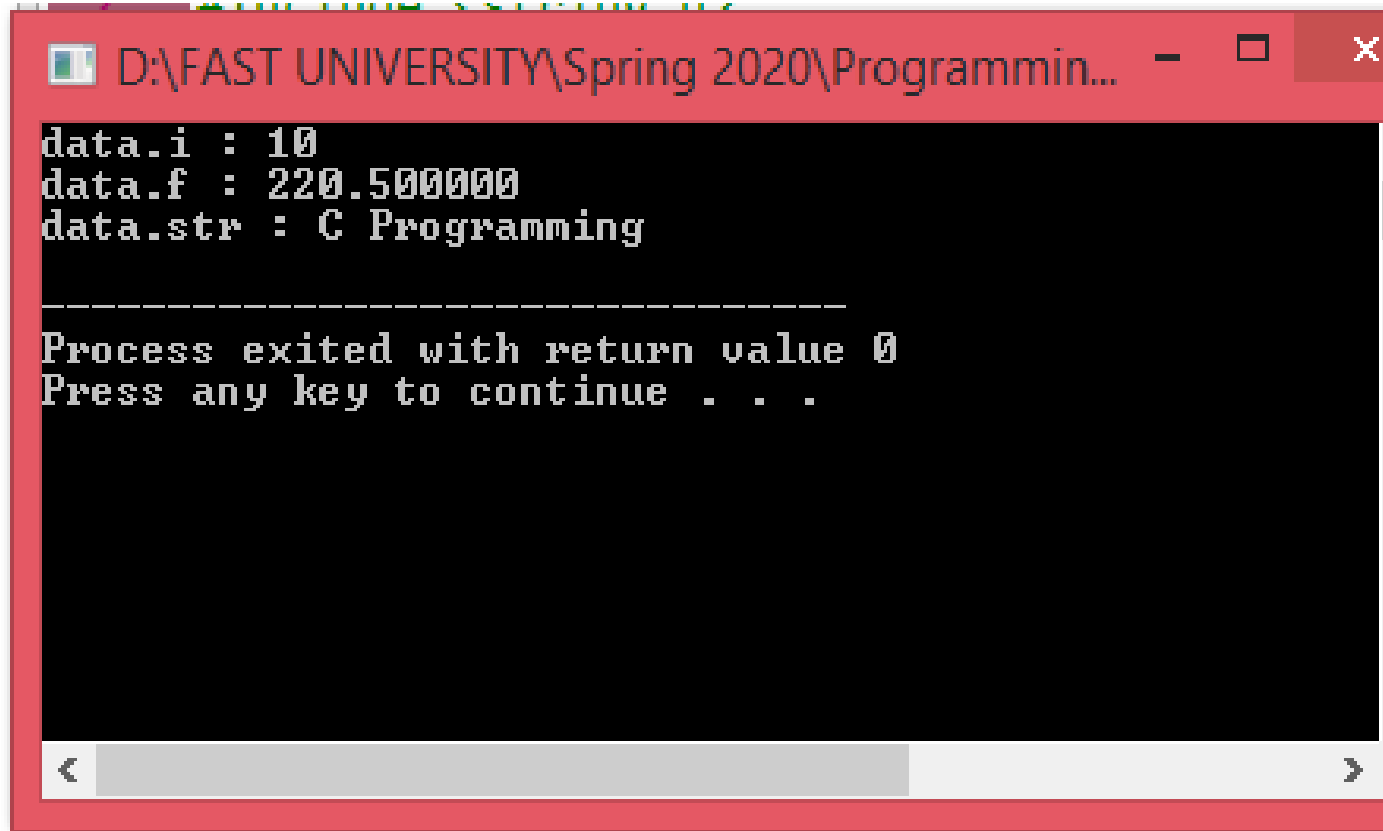
    union Data data;

    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;
```



```
D:\FAST UNIVERSITY\Spring 2020\Programmin...
data.i : 10
data.f : 220.500000
data.str : C Programming

-----
Process exited with return value 0
Press any key to continue . . .
```

Bitwise Operators

- All data represented internally as sequences of bits
 - Each bit can be either 0 or 1
 - Sequence of 8 bits forms a byte
 - The bitwise operators are summarized below

Operator		Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are <i>both</i> 1.
	bitwise inclusive OR	The bits in the result are set to 1 if <i>at least one</i> of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if <i>exactly one</i> of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent when the left operand is negative.
~	one's complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Bitwise Operators (Truth Tables)

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 10.8 | Results of combining two bits with the bitwise AND operator &.

Bit 1	Bit 2	Bit 1 Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 10.11 | Results of combining two bits with the bitwise inclusive OR operator |.

Bitwise Operators (Truth Tables)

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 10.12 | Results of combining two bits with the bitwise exclusive OR operator ^.

Bitwise assignment operators	
&■	Bitwise AND assignment operator.
■	Bitwise inclusive OR assignment operator.
^■	Bitwise exclusive OR assignment operator.
<<■	Left-shift assignment operator.
>>■	Right-shift assignment operator.

Fig. 10.14 | The bitwise assignment operators.

Bitwise AND operator &

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100

& 00011001

00001000 = 8 (In decimal)

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Bitwise OR operator |

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

```
00001100
| 00011001
-----
```

00011101 = 29 (In decimal)

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

Bitwise XOR (exclusive OR) operator ^

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

00010101 = 21 (In decimal)

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

Bitwise complement operator \sim

For any integer n , bitwise complement of n will be $-(n+1)$. To understand this, you should have the knowledge of 2's complement

2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

Decimal	Binary	2's complement
0	00000000	$-(11111111+1) = -00000000 = -0(\text{decimal})$
1	00000001	$-(11111110+1) = -11111111 = -256(\text{decimal})$
12	00001100	$-(11110011+1) = -11110100 = -244(\text{decimal})$
220	11011100	$-(00100011+1) = -00100100 = -36(\text{decimal})$

Bitwise complement operator ~

```
#include <stdio.h>
int main()
{
    printf("Output = %d\n",~35);
    printf("Output = %d\n",~-12);
    return 0;
}
```

Shift Operators

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

Shift Operators

- Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 = 11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) = 3392(In decimal)
```


Shift Operators

```
#include <stdio.h>
int main()
{
    int num=212, i;
    for (i=0; i<=2; ++i)
        printf("Right shift by %d: %d\n", i, num>>i);

    printf("\n");

    for (i=0; i<=2; ++i)
        printf("Left shift by %d: %d\n", i, num<<i);

    return 0;
}
```

Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848