**Union**

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.

**Structure** and **union** both are user defined data types which contains variables of **different** data types. ... In **union**, the total memory space allocated is equal to the member with largest size. All other members share the same memory space.

**Unions** are used to save only one type of data at a time. If a value is reassigned the old value is overwritten and cannot be accessed. In your example int ,float and char members can all have different values at any time when used as a **struct**. Its not the case in **union**.

**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 [union tag] {

member definition;

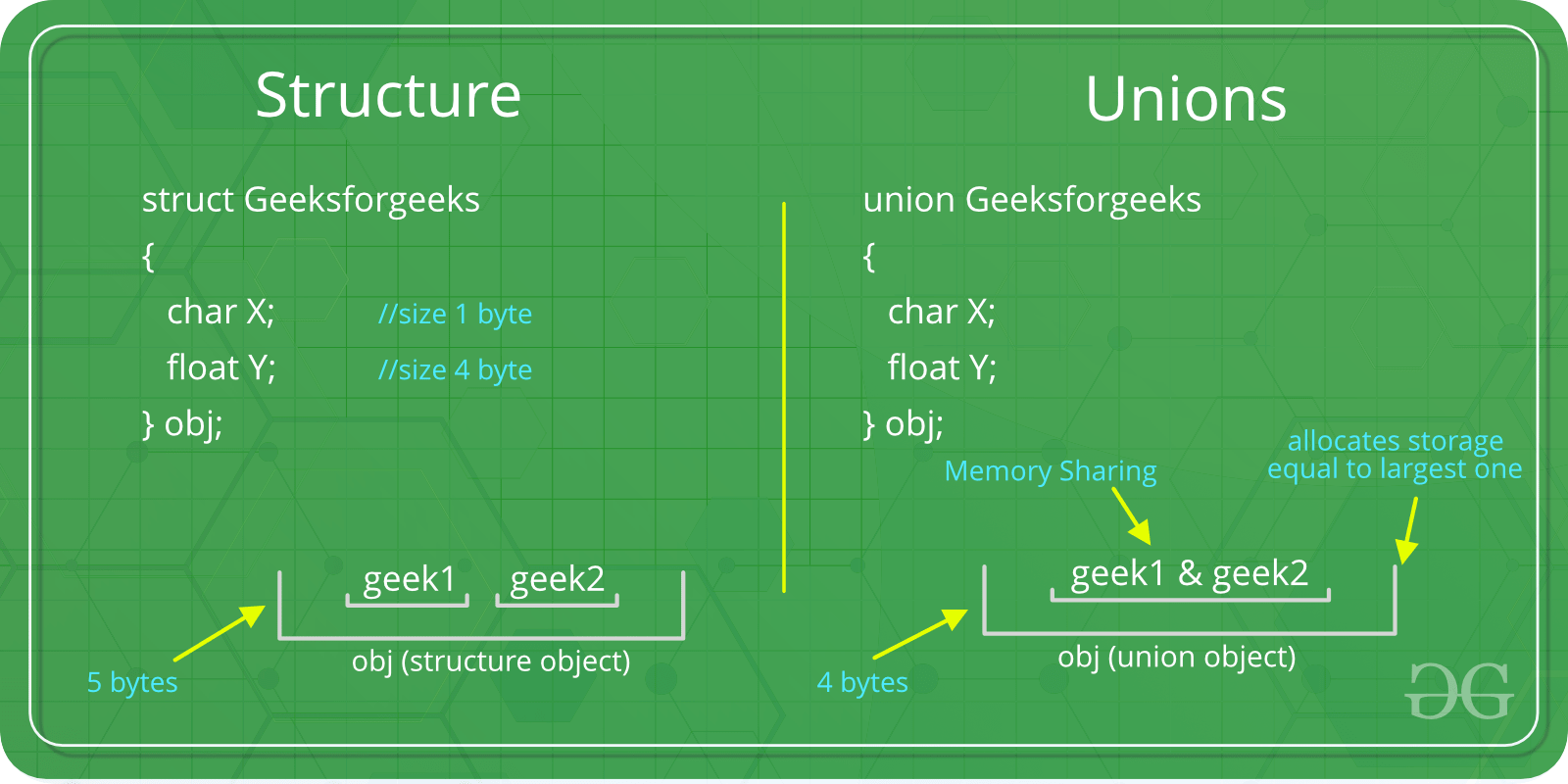
member definition;

...

member definition;

} [one or more union variables];

Union in C

[Structures](https://www.geeksforgeeks.org/structures-c/), union is a user defined data type. In union, all members share the same memory location. 

For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

|  |
| --- |
| #include <stdio.h>    // Declaration of union is same as structures  union test {      int x, y;  };    int main()  {      // A union variable t      union test t;        t.x = 2; // t.y also gets value 2      printf("After making x = 2:\n x = %d, y = %d\n\n",             t.x, t.y);        t.y = 10; // t.x is also updated to 10      printf("After making y = 10:\n x = %d, y = %d\n\n",             t.x, t.y);      return 0;  } |

**Output:**

After making x = 2:

x = 2, y = 2

After making y = 10:

x = 10, y = 10

**How is the size of union decided by compiler?**  
Size of a union is taken according the size of largest member in union.

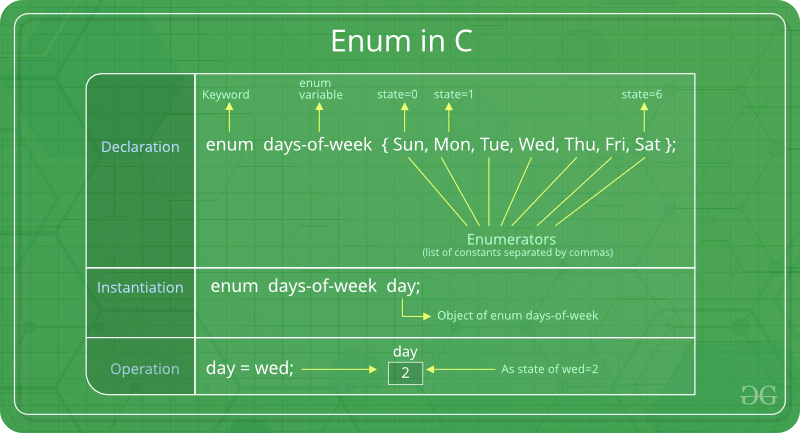
|  |
| --- |
| #include <stdio.h>    union test1 {      int x;      int y;  } Test1;    union test2 {      int x;      char y;  } Test2;    union test3 {      int arr[10];      char y;  } Test3;    int main()  {      printf("sizeof(test1) = %lu, sizeof(test2) = %lu, "             "sizeof(test3) = %lu",             sizeof(Test1),             sizeof(Test2), sizeof(Test3));      return 0;  } |

**Output:**

sizeof(test1) = 4, sizeof(test2) = 4, sizeof(test3) = 40

Enumeration (or enum) in C

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.



enum State {Working = 1, Failed = 0};

The keyword ‘enum’ is used to declare new enumeration types in C

Variables of type enum can also be defined. They can be defined in two ways:

// In both of the below cases, "day" is

// defined as the variable of type week.

enum week{Mon, Tue, Wed};

enum week day;

// **Or**

enum week{Mon, Tue, Wed}day;

|  |
| --- |
| // An example program to demonstrate working  // of enum in C  #include<stdio.h>    enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};    int main()  {      enum week day;      day = Wed;      printf("%d",day);      return 0;  } |

Output:

2

In the above example, we declared “day” as the variable and the value of “Wed” is allocated to day, which is 2. So as a result, 2 is printed.

Another example of enumeration is:

|  |
| --- |
| // Another example program to demonstrate working of enum in C  #include<stdio.h>    enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,            Aug, Sep, Oct, Nov, Dec};    int main()  {     int i;     for (i=Jan; i<=Dec; i++)        printf("%d ", i);       return 0;  } |

Output:

0 1 2 3 4 5 6 7 8 9 10 11

In this example, the for loop will run from i = 0 to i = 11, as initially the value of i is Jan which is 0 and the value of Dec is 11.

**Interesting facts about initialization of enum.**  
**1.** Two enum names can have same value. For example, in the following C program both ‘Failed’ and ‘Freezed’ have same value 0.

|  |
| --- |
| #include <stdio.h>  enum State {Working = 1, Failed = 0, Freezed = 0};    int main()  {     printf("%d, %d, %d", Working, Failed, Freezed);     return 0;  } |

Output:

1, 0, 0

**2.** If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0. For example, in the following C program, sunday gets value 0, monday gets 1, and so on.

|  |
| --- |
| #include <stdio.h>  enum day {sunday, monday, tuesday, wednesday, thursday, friday, saturday};    int main()  {      enum day d = thursday;      printf("The day number stored in d is %d", d);      return 0;  } |

Output:

The day number stored in d is 4

**3.** We can assign values to some name in any order. All unassigned names get value as value of previous name plus one.

|  |
| --- |
| #include <stdio.h>  enum day {sunday = 1, monday, tuesday = 5,            wednesday, thursday = 10, friday, saturday};    int main()  {      printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,              wednesday, thursday, friday, saturday);      return 0;  } |

Output:

1 2 5 6 10 11 12

**4.** The value assigned to enum names must be some integeral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value.