

# Unions in C

By

**Prof. Muhammad Iqbal Bhat**

Department of Higher Education  
Government Degree College Beerwah

# Topics:

1

What are  
Unions

2

Allowed  
Union  
Operations

3

Declaring and  
Accessing  
Unions

4

Union vs  
Structure

# What are Unions?



Like a structure, a union is a derived data type, but its members share the same memory.



Unions are a C language construct that allows the programmer to define a data type that can hold different types of data at different times.



This can be useful when you need to store data of different types in the same memory location.



Unions are similar to structs in that they are composed of a collection of named data elements, but with the key difference that only one of the member variables can be accessed at a time.



This means that accessing a union's members can be a bit trickier than accessing a struct's members, since you need to make sure you're accessing the right one at the right time.

# Allowed unions Operations



assigning a union to another union of the same type,



taking a union variable's address (&),

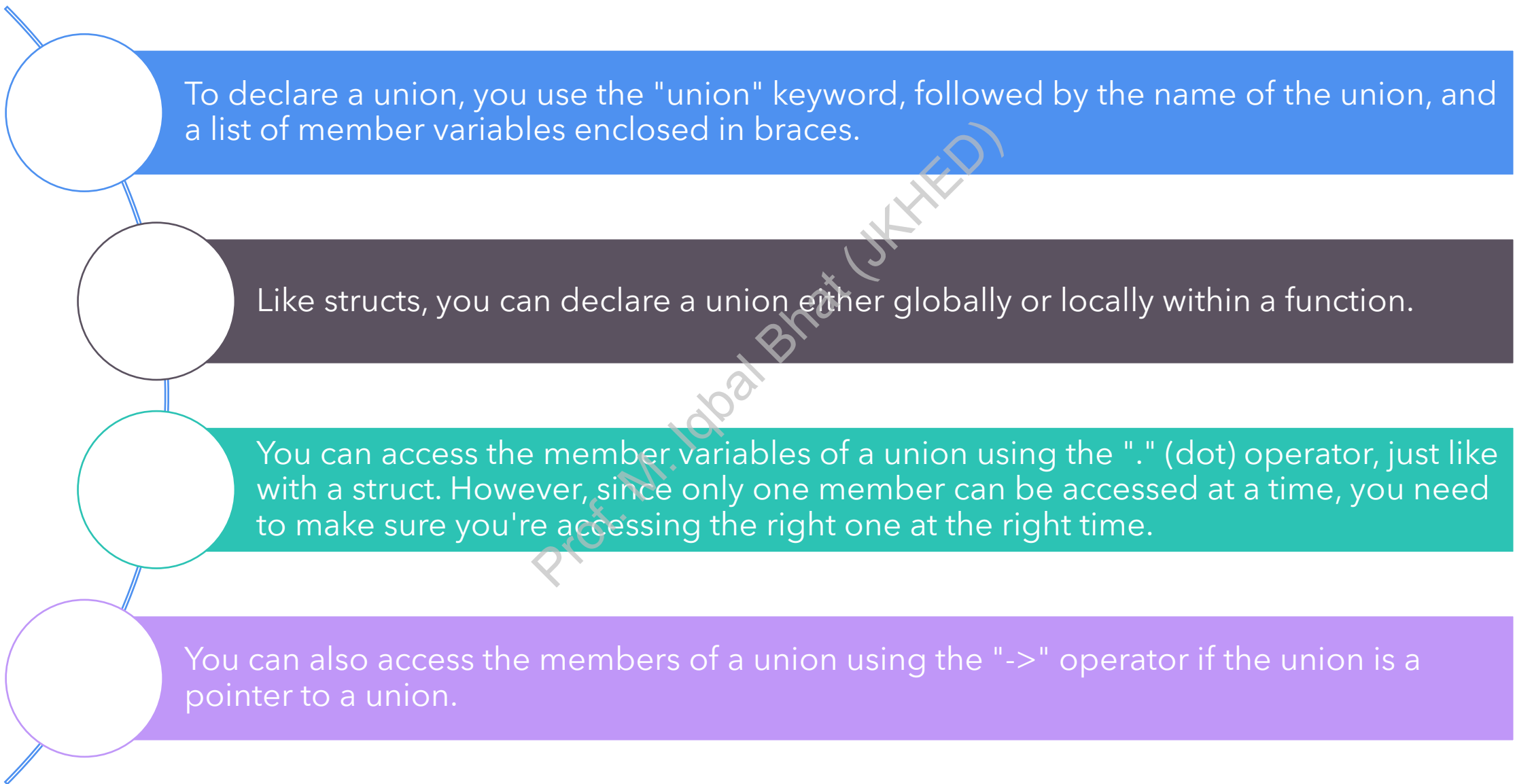


accessing union members via the structure member operator (.) and the structure pointer operator (->), and



zero-initializing the union.

# Declaring and Accessing Unions:



To declare a union, you use the "union" keyword, followed by the name of the union, and a list of member variables enclosed in braces.

Like structs, you can declare a union either globally or locally within a function.

You can access the member variables of a union using the "." (dot) operator, just like with a struct. However, since only one member can be accessed at a time, you need to make sure you're accessing the right one at the right time.

You can also access the members of a union using the "->" operator if the union is a pointer to a union.

# Syntax for Creating Unions:

The syntax for creating a Union in C is

```
union <union_name>
{
    data_type member1;
    data_type member2;
    ...
    data_type memberN;
};
```

```
union number {
    int x;
    double y;
};
```

# Defining Variables of Union Types

Following is the syntax to declare variables of a Union

```
union number {  
    int x;  
    double y;  
};
```

```
union number value;
```

## Initializing unions in Declarations

You can initialize a union in a declaration with a value of the union's first member type.

```
union number value = {10};
```

If you initialize the object with a double, as in

```
union number value = {1.43};
```

C will truncate the initializer value's floating-point part—some compilers will issue a warning about this.

# Size of Unions:

The size of a union is equal to the size of its largest member variable.

This is because the union must be large enough to accommodate the largest member variable.

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

```
union myUnion {  
    char c;  
    short s;  
    int i;  
};
```

```
int main() {  
    union myUnion u = {'A'};  
    printf("Size of myUnion: %lu\n", sizeof(u));  
    return 0;  
}
```



# Accessing Unions Members:

Only one member of a union can be accessed at a time.

The most recently assigned member is the one that can be accessed.

Accessing a member of a union that was not most recently assigned is undefined behavior.

It's a good practice to keep track of which member of the union is currently valid.

```
#include <stdio.h>
union myUnion {
    int i;
    float f;
};
int main() {
    union myUnion u;
    u.i = 42;
    printf("value of u.i: %d\n", u.i); // value of u.i: 42
    u.f = 3.14;
    printf("value of u.f: %f\n", u.f); // value of u.f: 3.140000
    printf("value of u.i: %d\n", u.i); // value of u.i: 1078523331
    return 0;
}
```

# Union vs Structure

Feature	Union	Structure
Memory layout	All members share the same memory location	Each member has its own memory location
Accessing	Only one member can be accessed at a time	All members can be accessed individually
Size	Size is equal to the largest member	Size is the sum of the sizes of all the members
Initialization	Only the first member is initialized by default	All members can be initialized separately or together
Usage	Useful when you need to store different types of data together	Useful when you need to group related data together

# Examples of Unions:

```
#include <stdio.h>

// number union definition
union number {
    int x;
    double y;
};

int main(void) {
    union number value; // define a union variable

    value.x = 100; // put an int into the union
    puts("Put 100 in the int member and print both members:");
    printf("int: %d\ndouble: %.2f\n\n", value.x, value.y);

    value.y = 100.0; // put a double into the same union
    puts("Put 100.0 in the double member and print both members:");
    printf("int: %d\ndouble: %.2f\n\n", value.x, value.y);
}
```

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

```
union myUnion {  
    int i;  
    float f;  
    char c;  
};
```

```
int main() {  
    union myUnion *pu;  
    union myUnion u;  
  
    pu = &u;  
    pu->i = 42;  
    printf("value of u.i: %d\n", pu->i);  
    pu->f = 3.14;  
    printf("value of u.f: %f\n", pu->f);  
    pu->c = 'a';  
    printf("value of u.c: %c\n", pu->c);  
    printf("value of u.i after assigning 'a': %d\n", pu->i);  
    return 0;  
}
```



Prof. M. Iqbal Bhat (JKHED)

**Questions?**

