Unit VII – Part 03
(Union in C)

Lecture Slide

AS2023

Objectives

By the end of this session, students will be able to:

- Define Union
- How to Create Union?
- How to access Union's members?
- Differences between Structure and Union
- Type Conversion: How convert from One Type to another?

Introduction to Union

- A ***union*** is a user-defined type similar to structs in *C* except for one key difference.
- Structs allocate enough space to store all its members
- Whereas Unions allocate the space equivalent to the size of the largest member

How to define a Union?

- We use the `union` keyword to define unions.
- Here's an example:

```
union car {
      char name[50];
      int price;
};
```

- The above code defines a derived type `union car`.

Create union variables

- When a union is defined, it creates a user-defined type. However, no memory is allocated.
- To allocate memory for a given union type and work with it, we need to create variables.

```
union car {
        char name[50];
        int price;
};
int main() {
        union car car1,
        car2, *car3;
        return 0;
}
```

```
union car {
        char name[50];
        int price;
} car1, car2, *car3;
```

Access Members of a Union

- We use the . operator to access members of a union using normal variable.
- To access members using pointer variables, we use the `->` operator.
- In the above example,
  - To access `price` for `car1, car1.price` is used.
  - To access `price` using `car3`, either `(*car3).price` or `car3->price` can be used.

Differences between Unions and Structures

```
#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 = %lu bytes\n", sizeof(uJob));
        printf("size of structure = %lu bytes",sizeof(sJob));
        return 0;
}
```

Output Sample:
size of union = 32
size of structure = 40

Why this difference in the size of variables?

- Here, the size of `sJob` is 40 bytes because
  - the size of `name[32]` is 32 bytes
  - the size of `salary` is 4 bytes
  - the size of `workerNo` is 4 bytes
- However, the size of `uJob` is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, `(name[32]),` is 32 bytes.

- With a union, all members share the **same memory**.

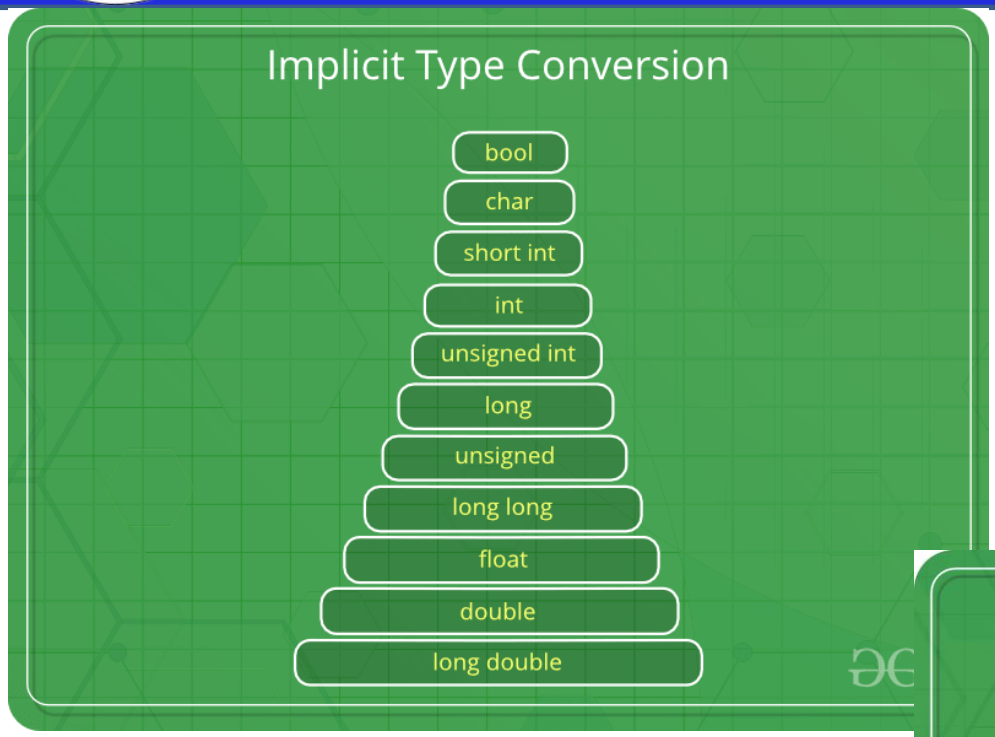
Example: Union

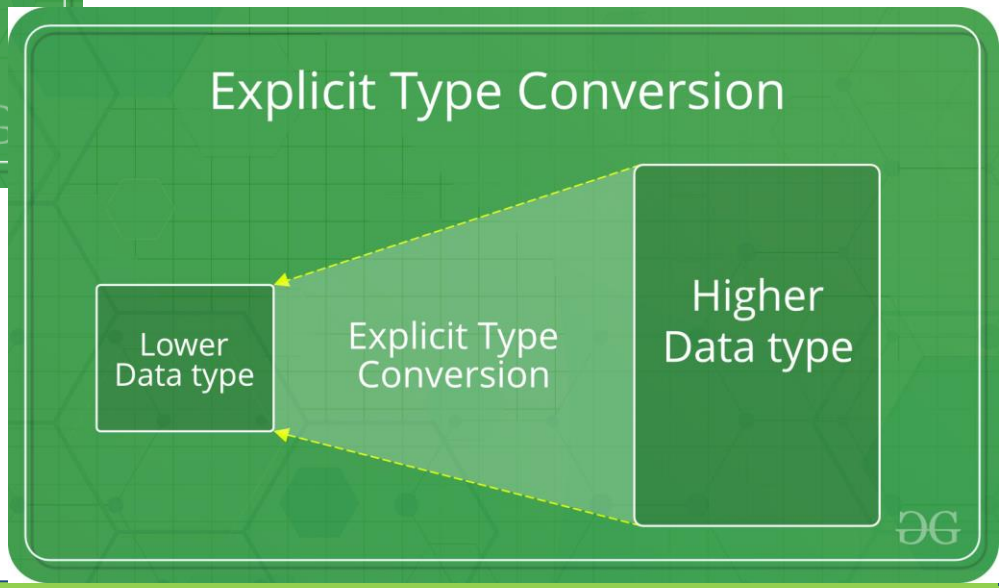
```
#include <stdio.h>
union Job {
        float salary;
        int workerNo;
} j;
int main() {
        j.salary = 12.3;
        // when j.workerNo is assigned a value,
        // j.salary will no longer hold 12.3
        j.workerNo = 100;
        printf("Salary = %.1f\n", j.salary);
        printf("Number of workers = %d", j.workerNo);
        return 0;
}
```

Output Sample:
Salary = Garbage Value
Number of workers = 100

Type Conversion
(Conversion of Data Types)

Types of Type Conversion





Implicit Type Conversion

- Also known as '***automatic type conversion***'.
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.
  - It is possible for implicit conversions to lose information.

```
bool -> char -> short int -> int -> unsigned int
-> long -> unsigned -> long long -> float ->
double -> long double
```

Example of Type Implicit Conversion

```
// An example of implicit conversion
#include<stdio.h>
int main() {
    int x = 10;    // integer x
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Output :
x = 107, z = 108.000000

Explicit Type Conversion

- This process is also called type casting and it is user defined.
- Here the user can type cast the result to make it of a particular data type.

- The syntax in C:

```
(data type) expression
```

Example for Explicit Type Conversion

```
// C program to demonstrate explicit type casting
#include<stdio.h>
int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

Output :
sum = 2

Advantages of Type Conversion

- Advantages of Type Conversion
  - This is done to take advantage of certain features of type hierarchies or type representations.
  - It helps us to compute expressions containing variables of different data types.
  - To reduce the data loss.

Thank you