# C++ Unions

- In C++, a union is a user-defined data type that can hold members of different data types. The size of the union will always be equal to the size of the largest member of the union. All the smaller elements can store the data in the same space without any overflow.
  - Unlike structures, all members of a union are stored in the same memory location.
  - Since all members share the same memory, changing the value of one member overwrites the value of the others.

```
#include <iostream>
using namespace std;

union A {
    int x;
    char y;
};

union B {
    int arr[10];
    char y;
};

int main()
{
    // Finding size using sizeof() operator
    cout << "Sizeof A: " << sizeof(A) << endl;
    cout << "Sizeof B: " << sizeof(B) << endl;

    return 0;
}
```

**Output**

Sizeof A: 4

Sizeof B: 40

The above union's memory can be visualized as shown:

**Union Declaration**

A union is declared similarly to a structure. Provide the name of the union and define its member variables:

*union union_Name {*
*type1 member1;*
*type2 member2;*

*..*
*};*

After declaration of a union then create a variable of union like below:

*union_name variable_name;*

We can also declare a variable at the declaration of union.

*union union_Name {*
*type1 member1;*
*type2 member2;*
*.*
*.*
*}variable_name;*

```
#include <iostream>
using namespace std;

union Student {
    int rollNo;
    float height;
    char firstLetter;
};

int main(){
    Student data;

    data.rollNo = 21;
    cout << data.rollNo << endl;

    data.height = 5.2;
    cout << data.height << endl;

    data.firstLetter = 'N';
    cout << data.firstLetter << endl;

    return 0;
}
```

**Output**

21

5.2

N

**Nested Union**

A union can be defined within a structure, class or another union, known as a nested union. This approach helps efficiently organize and access related data while sharing memory among the union's members.

**Syntax:**

*union name1 {*
*// Data members of Name1*
*union name2 {*
*// Data members of Name2*
*} inner;*
*};*

Accessing members of union using dot(.) operator:

*name1 obj;*
*obj.inner.member;*

**Example**:

```
#include <iostream>
using namespace std;

// Define a structure containing a nested union
struct Employee{
    char name[50];
    int id;

    // Nested union
    union Pay {
        float hourlyRate;
        float salary;
    } payment;
};

int main(){
    Employee e1;
    e1.id = 101;

    // Access nested union member using dot operator
    e1.payment.hourlyRate = 300.0;

    cout << "Employee ID: " << e1.id << "\n";
    cout << "Hourly Rate: Rs " << e1.payment.hourlyRate << endl;

    // You can also assign salary if needed
    e1.payment.salary = 50000.0;
    cout << "Salary: Rs " << e1.payment.salary << endl;
```

```
    return 0;
}
```

**Output**

Employee ID: 101

Hourly Rate: Rs 300

Salary: Rs 50000

**Anonymous Unions**

An anonymous union is a union without a name whose members can be accessed directly and behave like members of the outer structure or union.

```
#include <iostream>
using namespace std;

struct Employee{
    int id;

    // Anonymous union
    union {
        float hourlyRate;
        float salary;
    };
};

int main(){
    Employee e1;
    e1.id = 101;

    // Access anonymous union member
    e1.hourlyRate = 300;

    cout << "Employee ID: " << e1.id << "\n";
    cout << "Hourly Rate: Rs " << e1.hourlyRate << endl;

    // Only one member is active at a time
    e1.salary = 50000;
    cout << "Salary: Rs " << e1.salary << endl;

    return 0;
}
```

**Output**

Employee ID: 101

Hourly Rate: Rs 300

Salary: Rs 50000

## Applications of Union in C++

- Used for memory efficiency, especially in embedded systems
- Allows different data types to share the same memory, reducing footprint
- Can map hardware registers with multiple interpretations
- Suitable for data that takes one of several types, but never simultaneously