**Union**

# Union in C

* union is a user defined data type
* In union, all members share the same memory location
* Only one Member will be active at a time.

Suppose we are accessing one of the data member of union then

we cannot access other data member since we can access single

data member of union because each data member shares same memory

# Union definition

union <union\_name> {

data\_type memeber1;

data\_type memeber2;

.......

};

# Size of union

Size of a union is taken according the size of largest member in union

#include <stdio.h>

union test1 { int x; int y; };

union test2 { int x; char y; };

union test3 { int arr[10]; char y; };

int main() {

printf ("sizeof(test1) = %d, sizeof(test2) = %d,"

"sizeof(test3) = %d", sizeof(union test1),

sizeof(union test2), sizeof(union test3));

return 0;

}

Output: Copy

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

# Pointers to unions ?

we can have pointers to unions and can access members using arrow operator (->)

#include <stdio.h>

union test { int x; char y; };

int main() {

union test p1;

p1.x = 65;

union test \*p2 = &p1; // p2 is a pointer to union p1

printf("%d %c", p2->x, p2->y); // Accessing union members using pointer

return 0;

}

Output:

65 A

# What are applications of union ?

useful where we want to use same memory for two or more members

**Example**

We want to implement a binary tree data structure where each leaf node has a double data value, while each internal node has pointers to two children, but no data

struct NODE {

struct NODE \*left;

struct NODE \*right;

double data;

};

structure requires 16 bytes, with half the bytes wasted for each type of node

if we declare a node as following, then we can save space.

struct NODE {

bool is\_leaf;

union {

struct {

struct NODE \*left;

struct NODE \*right;

} internal;

double data;

} info;

};

# Anonymous Union in C

also known as unnamed union

// like anonymous structure, anonymous union is also similar

// Anonymous union declaration

union { char alpha; int num; };

* Since there is no names, direct objects(or variables) of them are not created and we use them in nested structure or unions
* Since there is no variable and no name, we can directly access members

// C Program to demonstrate working of anonymous union

#include <stdio.h>

struct Scope {

// Anonymous union

union {

char alpha;

int num;

};

};

int main() {

struct Scope x, y;

x.num = 65;

y.alpha = 'A';

// Note that members of union are accessed directly

printf("y.alpha = %c, x.num = %d", y.alpha, x.num);

return 0;

}

Ouput

y.alpha = A, x.num = 65

* **This is a C only feature**
* Anonymous Unions and Structures are NOT part of C++ 11 standard, but most of the C++ compilers support them
* The C++ implementations don’t allow to anonymous struct/union to have private or protected members, static members, and functions

# Difference between structure and union

|  |  |  |
| --- | --- | --- |
|  | Structure | Union |
| Keyword | struct | union |
| Size | >= sum of size of its members | Equal to the size of largest member |
| Memory | Each member is allocated it’s own memory | Memory allocated is shared by individual members of union |
| Value Altering | Altering the value of a member will not affect other members of the structure | Altering the value of a member will alter other member values |
| Accessing Members | Individual member can be accessed at a time | Only one member can be accessed at a time |
| Initializations of members | Several members of a structure can be initialized at once | Only the first member of a union can be initialized |

# References

<https://www.geeksforgeeks.org/union-c/>

<https://www.geeksforgeeks.org/difference-structure-union-c/>

<https://www.geeksforgeeks.org/g-fact-38-anonymous-union-and-structure/>

<https://en.cppreference.com/w/c/language/union>

<https://en.cppreference.com/w/cpp/language/union>

Chapter 10 | Compound Types: Enums and Structs <https://www.learncpp.com/>

# END