Structure and Union

INDEX

Table of Contents

[Class vs Struct vs Union in C++ 2](#_Toc142652055)

[Class 3](#_Toc142652056)

[Size of Class 3](#_Toc142652057)

[Struct 4](#_Toc142652058)

[Use case 4](#_Toc142652059)

[Designated Initialization 5](#_Toc142652060)

[Size of Structure 6](#_Toc142652061)

[Structure pointer 7](#_Toc142652062)

[Union 8](#_Toc142652063)

[Use case 8](#_Toc142652064)

[Size of Union 9](#_Toc142652065)

[Type punning 9](#_Toc142652066)

[Anonymous Union 10](#_Toc142652067)

[Structure Alignment and Padding in C++ 11](#_Toc142652068)

[Reason for Alignment 12](#_Toc142652069)

# Class vs Struct vs Union in C++

|  |  |  |
| --- | --- | --- |
| **Class** | **Struct** | **Union** |
| Members of a class are private by default. | Members of a structure are public by default. | Members of a structure are public by default. |
| It is normally used for data abstraction and further inheritance. | It is normally used for the grouping of data | It can be used where the amount of memory used is a key factor. |
| The size of a class is determined by the sizes of its **member variables**, any additional memory required for alignment and **padding**, and the compiler's **memory layout** considerations. | The size is the sum of the sizes of its **individual members**, possibly **padded** for alignment considerations. | The size is determined by the **largest data type** among its members, as all members share the same memory space |

Structure is just like class; one difference is default access specifier. In the below example that struct have same features as class like constructor, functions.

A white paper with black text

Description automatically generated

## Class

### Size of Class

A screen shot of a computer code

Description automatically generated

OP



## Struct

**Remember**

* When we declare a structure or union, we declare a new data type suitable for our purpose. So, **we cannot initialize values as it is not a variable declaration but a data type declaration**.

### Use case

* Store a point in a 2D coordinate system which has two members x and y, which represent coordinate.
* Store student information at one place.
* Store information of webpage, URL, no of views on the page or content of page in the form of string.

A computer screen shot of a program code

Description automatically generated

OP



### Designated Initialization

A black background with white text

Description automatically generated

OP



**Remember**

If we don’t initialize member and accessing members, then we can get some random value. But if we do designated Initialization for some value then other uninitialized value get default value 0.

A computer screen with white text

Description automatically generated

OP



### Size of Structure

A computer screen with white text

Description automatically generated

OP

Expected output (for a typical 32-bit architecture):



Expected output (for a typical 64-bit architecture):



**Remember**

* A structure cannot contain a member of its own type because if this is allowed then it becomes impossible for compiler to know size of such struct. Although a pointer of same type can be a member because pointers of all types are of same size and compiler can calculate size of struct

**A black background with white text

Description automatically generated**

### Structure pointer

**Remember**

The arrow operator -> is used to dereference the pointer and access its members.

A computer screen shot of a black background

Description automatically generated

OP



## Union

### Use case

* Type punning, it is a common term used for circumventing the formal type of system in programming languages. We basically convert one type into the other type without any explicit typecasting and we get the internal representation of that type.

**Remember**

* It allocates memory equal to the largest data type inside it.

Example to show union use same memory location and overwrite for all its members.

A screen shot of a computer program

Description automatically generated

Total size: 4 bytes

OP

A black screen with white text

Description automatically generated

### Size of Union

A computer screen with white and green text

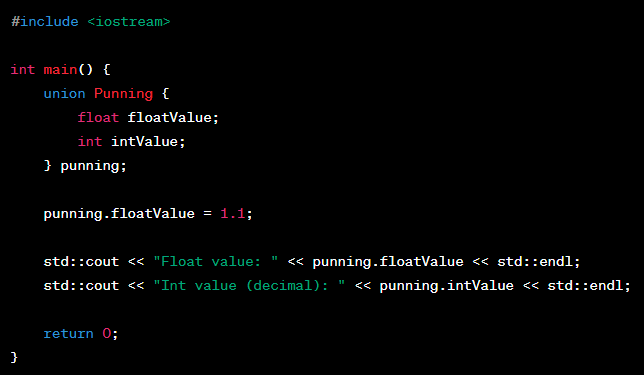
Description automatically generated

OP

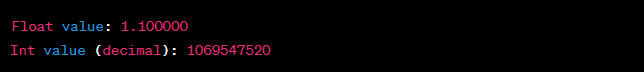


### Type punning

The value obtained when performing type punning using unions to interpret a floating-point value as an int can vary across different platforms, compilers, and optimization settings. (Int value can differ)



OP



Anonymous Union

Using anonymous union to use one type for multiple purposes.

* A Node structure that can be used for both Doubly Linked List and Binary Tree.

A computer screen with white text

Description automatically generated

# Structure Alignment and Padding in C++

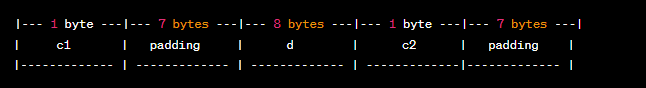
Structure alignment and padding are closely related concepts. Alignment ensures that **data is accessed efficiently**, and padding is the **extra space inserted between structure members** to achieve that alignment.

**Remember**

* A structure has alignment requirements same as its largest member’s requirement.
* Alignment requirements are machine or compiler specific, so we can’t read the size required by a structure on a random machine until unless we know individual data type alignment requirement.
* Largest member is **double** in this example. Double require 8-byte alignment requirement. So, 7-byte padded after c1 and c2.

A black background with a black square

Description automatically generated with medium confidence



Total size: 24 bytes

Largest member is **int** in this example. Int require 4-byte alignment requirement. So, 3-byte padded after c1 and c2.

A black background with a black square

Description automatically generated with medium confidence

A black background with white text

Description automatically generated

Total size: 12 bytes

**Remember**

By now, it may be clear that padding is unavoidable. There is a way to minimize padding. The programmer should declare the structure members in their increasing/decreasing order of size.

A black background with a black square

Description automatically generated with medium confidence

A black background with white text

Description automatically generated

Total size: 8 bytes

Reason for Alignment(Why wasting memory and doing alignment)

* Memory is byte addressable, but CPU read memory in the form of words, In modern computing architectures:
  + 32-bit machines use 4 bytes as their word size.
  + 64-bit machines use 8 bytes as their word size.

CPU read 4 or 8 bytes at a time to save the CPU cycle to read memory.

* If data is not aligned, then it might across multiple words. In the given example we can see that firstly CPU read 4 byte which contains c1, c2 and 2 bytes of i and in another cycle, CPU read remaining 2 byte. It means CPU need two CPU cycle to read I variable. So that’s the reason have alignment so that we can read these things in one word.

A black background with a black square

Description automatically generated with medium confidence

A black screen with white text

Description automatically generated

* We can use for compact representation of structure.

**\_\_attribute\_\_((packed))** prevent the compiler from adding padding between members.

A black background with a black square

Description automatically generated with medium confidence

A black background with white text

Description automatically generatedTotal size: 6 bytes