C++, в чём отличие структур и классов

В стандартном C++ структуры должны удовлетворять определённому набору правил и соответствуют типу простых данных (plain old data type или POD).

Простая структура данных — в современных языках программирования высокого уровня тип данных, имеющий жёстко определённое расположение полей в памяти, не требующий ограничения доступа и автоматического управления. Переменные такого типа можно копировать простыми процедурами копирования участков памяти (memcpy в C). Противоположность — **управляемая структура данных**.

В спецификации нет четкой разницы между **struct** и **class**. Основное различие заключается в ожиданиях программистов, когда они читают ваш код через 2 года. Структуры часто предполагаются POD.

Рекомендации по оформлению кода**.**
class vs struct. Всегда используйте class, только если вы не создаете очень маленький и простой вид данных, для которого нужны лишь несколько публичных переменных и, возможно, конструктор для их инициализации.

Классы в C++ — это абстракция описывающая методы, свойства, ещё не существующих объектов.
Объекты — конкретное представление абстракции, имеющее свои свойства и методы. Созданные объекты на основе одного класса называются экземплярами этого класса. Эти объекты могут иметь различное поведение, свойства, но все равно будут являться объектами одного класса. В ООП существует три основных принципа построения классов:

1. **Инкапсуляция** — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.
2. **Наследование** — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.
3. **Полиморфизм** — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

STRUCT – это тип данных, который делит ***непрерывный*** кусок памяти в соответствии со спецификацией структуры. Структуры особенно полезны при ***сериализации файлов/десериализации***, поскольку структуру часто можно записать в файл дословно (т.е. получить указатель на структуру, использовать оператор SIZEOF для вычисления количества байтов для копирования, а затем переместить данные в или из структуры.)

Классы – это абстрактный тип данных, который пытается обеспечить скрытие информации. Внутри могут быть различные методы, временные переменные, переменные состояния и т.д. Классы используются для объявления классов с использованием полноценного механизма ООП.

Отличия:

- Члены ***структуры*** (или `union`) являются общедоступными (`public`) по умолчанию, члены ***класса*** являются закрытыми (`private`) по умолчанию.

- Наследование по умолчанию для структуры из другой структуры или класса является `public`. Тогда как `Default` наследование для класса из другой структуры или класса является закрытым.
- если не определяется конструктор в ***классе***, компилятор определит его, но в ***структуре***, если он не определён явно, компилятор ***не определяет*** конструктор.
- Еще одно главное отличие в том, что касается шаблонов: допускается использование ключевого слова `class`, когда вы определяете шаблон, но НЕ `struct`.
    - `template<class T> // OK`
    - `template<struct T> // ERROR, struct not allowed here`

Также обратите внимание, что стандарт C++ позволяет вам заранее объявить тип как `struct`, а затем использовать `class` при объявлении типа и наоборот. Кроме того, `std::is_class<Y>::value` является `true` для Y, являющимся `struct` и `class`, но `false` для `enum class`.

```
template <class T>
struct H
{
  T x;
  T y;
}
H<int> ddd;
```

А так со структурой нельзя.

```
template <struct T>
struct H
{
  T x;
  T y;
}
H<int> ddd;
```

Допустим, есть такой код:

```
struct C;
class C;
```

Чем, по-твоему, является C — классом или структурой?

C++20 (справочник по языку)
Существует три типа классов: структура, класс и объединение. Они объявляются с помощью ключевых слов `struct`, `class` и `union`. В следующей таблице показаны различия между этими тремя типами.

Управление доступом и ограничения для структур, классов и объединений

Структуры	**Классы**	**Объединения**
Ключевое слово для класса: `struct`	Ключевое слово для класса: `class`	Ключевое слово для класса: `union`
Доступ по умолчанию: `public` (открытый).	Доступ по умолчанию: `private` (закрытый).	Доступ по умолчанию: `public` (открытый).
Нет ограничений на использование	Нет ограничений на использование	Используется только один член за один раз

Классы и структуры являются конструкциями, в которых пользователь определяет собственные типы. Классы и структуры могут включать данные-члены и функции-члены, позволяющие описывать состояние и поведение данного типа.

В C++ структура совпадает с классом, за исключением того, что его члены `public` по умолчанию.

Классы и структуры в .NET Framework похожи. И те, и другие могут иметь поля, свойства и события. Они также могут иметь статические и нестатические методы. Отличием является то, что структуры являются ***типами значений***, а классы — ***ссылочными типами***.

Ссылочные типы (`ref`) можно создавать только в ***управляемой куче***, а не в стеке или в собственной куче. Экземпляры ***типов значений*** можно создавать в ***стеке или в управляемой куче***.

Расширения **`ref class`** или **`ref struct`** объявляют класс или структуру со *временем жизни объекта*, которое администрируется ***автоматически***. Когда объект становится недоступным или выходит за пределы области, память освобождается.

```
class_access ref class name modifier : inherit_access base_type {};
class_access ref struct name modifier : inherit_access base_type {};
```

Термин структура относится к организации членов объекта класса, структуры или типа объединения в памяти. В некоторых случаях структура четко определена спецификациями языка. Однако если класс или структура содержит определенные возможности языка C++, такие как виртуальные базовые классы, виртуальные функции, члены с разным уровнем управления доступом, ***компилятор может выбрать структуру самостоятельно***. Эта структура может сильно отличаться в зависимости от того, какие оптимизации выполняются, и во многих случаях объект может даже не занимать непрерывную область памяти.

Если класс или структура является простым типом и типом стандартной структуры — это тип POD (Plain Old Data, обычные старые данные). Распределение памяти для типов POD является ***непрерывным***, и адрес каждого члена выше, чем адрес члена, объявленного до него, что дает возможность выполнять побайтовое копирование и двоичный ввод-вывод для этих типов. Скалярные типы, такие как int, также являются типами POD. Типы POD, которые являются классами, могут содержать только типы POD в качестве нестатических членов данных.

https://docs.microsoft.com/ru-ru/cpp/cpp/trivial-standard-layout-and-pod-types?view=msvc-170

Структура или объединение, все нестатические данные-члены которого имеют типа POD, также обладает типом POD при соблюдении следующих условий:

- Нет объявленных пользователем конструкторов.
- Нет закрытых или защищенных нестатических данных-членов.
- Отсутствие базовых классов.
- Нет виртуальных функций.
- Нет нестатических данных-членов ссылочного типа.
- Нет определяемого пользователем оператора присвоения копирования.
- Нет определяемого пользователем деструктора.

Таким образом, можно рекурсивно создавать структуры POD и массивы, содержащие структуры и массивы POD.

https://docs.microsoft.com/ru-ru/cpp/standard-library/is-pod-class?view=msvc-170

POD типы

POD («Plain Old Data») могут обрабатываться, как C-структуры, т.е. копироваться с помощью **memcpy()**, инициализироваться с помощью **memset()**, и т.д. В C++98 определение POD было основано на наборе ограничений языковых конструкций, используемых при определении структуры:

```
// S - это POD
struct S { int a; };
// SS – это не POD
struct SS { int a; SS(int aa) : a(aa) { } };
struct SSS { virtual void f(); /* ... */ };
```

В C++11 `S` и `SS` являются «типами со стандартным расположением в памяти» (standard layout type) (a.k.a. POD) поскольку `SS` не содержит никакой «магии»: конструктор никак не влияет на расположение в памяти (поэтому инициализация с помощью `memcpy()` является возможной), и только лишь с инициализацией с помощью `memset()` будут проблемы, поскольку инициализация таким образом не будет обеспечивать инвариант. Однако, `SSS` будет содержать указатель на таблицу виртуальных функций `vptr`, и не может рассматриваться как «простая старая структура данных». В C++11 для работы с разными техническими аспектами, даются следующие определения: POD-типов, простых копируемых типов и типов со стандартным расположением в памяти. Определение POD-типа является рекурсивным:

- Если все члены и базовые типы являются POD-типами, то текущий тип является POD-типом
- Как и ранее
    - Отсутствие виртуальных функций
    - Отсутствие виртуальных базовых классов
    - Отсутствие ссылок
    - Отсутствие нескольких спецификаторов доступа

Наиболее важный аспект POD-типов в C++11 заключается в том, что добавление или удаление конструкторов не влияет на производительность или расположение объектов в памяти.