**C++ Tips**

Differences Between Class and Struct:

The only **difference** between a **struct and class** in **C++** is the default accessibility of member variables and methods. In a **struct** they are public; in a **class** they are private.

Heap and Stuck Differences:

**Stack** is used for static memory allocation and **Heap** for dynamic memory allocation, both stored in the computer's RAM. Variables allocated on the **stack** are stored directly to the memory and access to this memory is very fast, and it's allocation is dealt with when the program is compiled.

Nesneye Yönelik Programlama Dillerinin Özellikleri:

1. Abstraction (Soyutlama)
2. Sarmalama / Paketleme (Encapsulation)
3. Miras Alma (Inheritance)
4. Çok Biçimlilik (Polymorphism)

Why we need virtual destructor in C++ ?

If your derived class **destructor** is **virtual** then objects will be destrcuted in a order(firstly derived object then base ). If your derived class **destructor** is NOT **virtual** then only base class object will get deleted(because pointer is of base class "Base \*myObj"). So there will be memory leak for derived object.

Differences Between Vector and Array

Vectors

* do their own allocation and deallocation of memory
* resize automatically when expanding
* can be used where STL containers can, which includes many many algorithms and methods
* have many other methods that are useful

Arrays

* take less memory overall
* Are used in C code, so are used for reverse compatibility

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Internally, vectors use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

Instead, vector containers may allocate some extra storage to accommodate for possible growth, and thus the container may have an actual capacity greater than the storage strictly needed to contain its elements (i.e., its size). Libraries can implement different strategies for growth to balance between memory usage and reallocations, but in any case, reallocations should only happen at logarithmically growing intervals of size so that the insertion of individual elements at the end of the vector can be provided with amortized constant time complexity (see push\_back).

Therefore, compared to arrays, vectors consume more memory in exchange for the ability to manage storage and grow dynamically in an efficient way.

Compared to the other dynamic sequence containers (deques, lists and forward\_lists), vectors are very efficient accessing its elements (just like arrays) and relatively efficient adding or removing elements from its end. For operations that involve inserting or removing elements at positions other than the end, they perform worse than the others, and have less consistent iterators and references than lists and forward\_lists.

dynamic\_cast vs static\_cast

**static\_cast**: This is used for the normal/ordinary type conversion. This is also the cast responsible for implicit type conversion and can also be called explicitly. You should use it in cases like converting float to int, char to int, etc.

**dynamic\_cast**: This cast is used for handling polymorphism. You only need to use it when you're casting to a derived class. This is exclusively to be used in inheritence when you cast from base class to derived class.

**Regular Cast**: This is the most powerful cast available in C++ as it combines const\_cast, static\_cast and reinterpret\_cast. but it's also unsafe because it does not use dynamic\_cast. This is also called as C-style cast.

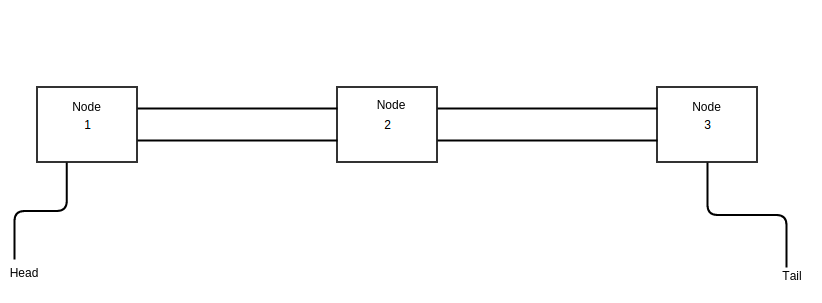
Other Available casts

**const\_cast**: can be used to remove or add const to a variable. This can be useful if it is necessary to add/remove constness from a variable.

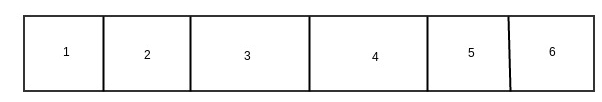
**reinterpret\_cast**: This is the trickiest to use. It is used for reinterpreting bit patterns and is extremely low level. It's used primarily for things like turning a raw data bit stream into actual data or storing data in the low bits of an aligned pointer.

Vector vs List

List stores elements at non contiguous memory location i.e. it internally uses a doubly linked list i.e.

[](https://thispointer.com/wp-content/uploads/2016/05/linkedlist.png)

Whereas, vector stores elements at contiguous memory locations like an array i.e.

[](https://thispointer.com/wp-content/uploads/2016/06/array.jpg)

* Insertion and Deletion in List is very efficient as compared to vector because to insert an element in list at start, end or middle, internally just a couple of pointers are swapped.
* Whereas, in vector insertion and deletion at start or middle will make all elements to shift by one. Also, if there is insufficient contiguous memory in vector at the time of insertion, then a new contiguous memory will be allocated and all elements will be copied there.
* So, insertion and deletion in list is much efficient than vector in c++.
* As List is internally implemented as doubly linked list, therefore no random access is possible in List. It means, to access 15th element in list we need to iterate through first 14 elements in list one by one.
* Whereas, vector stores elements at contiguous memory locations like an array. Therefore, in vector random access is possible i.e. we can directly access the 15th element in vector using operator []
* So, we can not use std::list with some of the STL algorithm that needs the random access operators like std::sort.
* Deleting or Inserting an element in List does not invalidate any iterator because during insertion and deletion no element is moved from its position only a couple pointers are changed.
* Whereas, in vector insertion and deletion can invalidate the iterators.

Set vs Map

The difference is set is used to store only keys while map is used to store key value pairs.

Conceptually, a set is a collection of things, whereas a map is a mapping of keys to values. A map stores keys sorted. std::map is an associative container storing pairs of key-values with unique keys. std::set is also an associative container that stores a sorter set of objects (or keys).

A map stores keys sorted. It maps keys to values. Usually it is implemented as a binary search tree (red-black tree) for keys. A set is a map where values are irrelevant. unordered\_map and unordered\_set (new in C++11) store keys unsorted and use hash table for search

When should we write our own assignment operator in C++?

The answer is same as Copy Constructor. If a class doesn’t contain pointers, then there is no need to write assignment operator and copy constructor. The compiler creates a default copy constructor and assignment operators for every class. The compiler created copy constructor and assignment operator may not be sufficient when we have pointers or any run time allocation of resource like file handle, a network connection..etc.

1) Do not allow assignment of one object to other object. We can create our own dummy assignment operator and make it private.

2) Write your own assignment operator that does deep copy.

Same is true for Copy Constructor.

<https://www.geeksforgeeks.org/assignment-operator-overloading-in-c/>