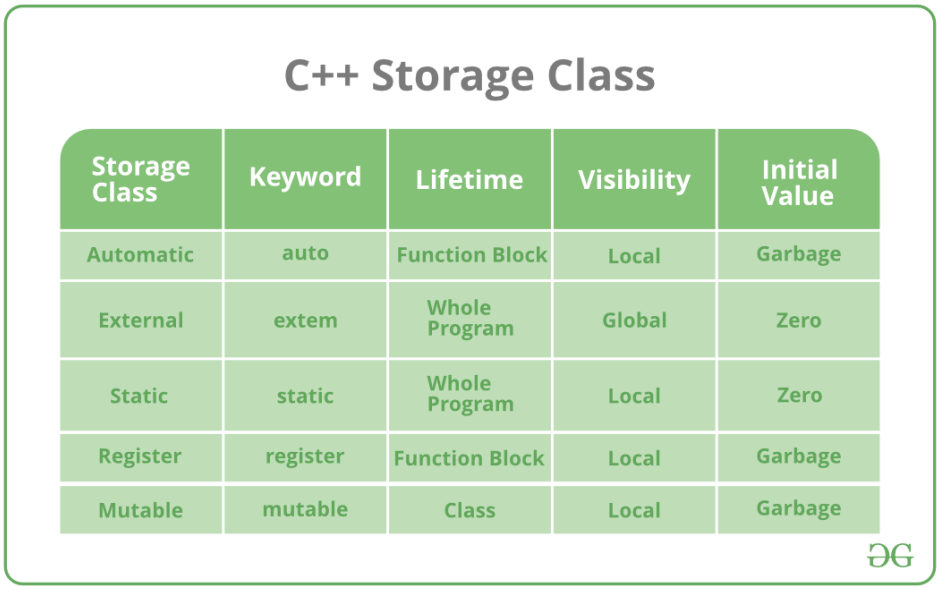
Page 2

30 October 2020 19:45

1. **Class -** Class is a blue print which reflects the entities attributes and actions. Technically defining a class is designing an user defined data type.
2. **Object -** An instance of a class.
3. **Inheritance -** Single, Multilevel, Multiple, Hierarchical and Hybrid.
4. **Storage Classes -** It tells the scope, life-time and visibility of a variable.
   a. Auto - It provides type inference capabilities, using which automatic deduction of the data type of an expression in a programming language can be done
   b. Extern - Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used



5. Pure Virtual Function - A virtual function with no function body and assigned with a value zero is called as pure virtual function. A class is abstract if it has at least one pure virtual function.
   virtual void show() = 0;
6. By default the members of struct are public and by default the members of the class are private
7. **Virtual destructors** are useful when you might potentially delete an instance of a derived class through a pointer to base class.
8. #define is a pre-processor token: the compiler will never see it.
   typedef is a compiler token
   #define should not be terminated with a semicolon, but typedef should be terminated with semicolon.

```
typedef char* ch;
ch a,b,c;  ==  char *a, *b, *c;
#define ch char*
ch a,b,c ; ==  char *a, b, c;
```

9. **Static Memory Allocation:**
   Memory is allocated for the declared variable by the compiler. The address can be obtained by using 'address of' operator and can be assigned to a pointer. The memory is allocated during compile time. Since most of the declared variables have static memory, this kind of assigning the address of a variable to a pointer is known as static memory allocation.
   Dynamic Memory Allocation:
   Allocation of memory at the time of execution (run time) is known as dynamic memory allocation. The functions calloc() and malloc() support allocating of dynamic memory. Dynamic allocation of memory space is done by using these functions when value is returned by functions and assigned to pointer variables.

10. Stack memory allocation is done at runtime.

11. A lock allows only one thread to enter the part that's locked and the lock is not shared with any other processes.
    A mutex is the same as a lock but it can be system wide (shared by multiple processes).
    A semaphore does the same as a mutex but allows x number of threads to enter, this can be used for example to limit the number of cpu, io or ram intensive tasks running at the same time.

12. Deleting a derived class object using a pointer of base class type that has a non-virtual destructor results in undefined behaviour.
    ```
    virtual ~base() {}
    ```

13. Destructors are not called automatically when the object goes out of scope. Instead we have to explicitly call the delete function to delete the memory assigned to the class.

14. Type Safe means that variables are statically checked for appropriate assignment at compile time.

15. Differences between a class and a struct in C++ are that structs have default public members and bases and classes have default private members and bases. Both classes and structs can have a mixture of public, protected and private members, can use inheritance and can have member functions.
16. C++ doesn't have a garbage collector, resources that we don't free ourselves will never be released back to the system, eventually making things grind to a halt.

17. **Volatile** - volatile is a hint to the implementation to avoid aggressive optimization involving the object because the value of the object might be changed by means undetectable by an implementation
    ```
    int some_int = 100;
    while(some_int == 100)
    {
      //your code
    }
    ```

When this program gets compiled, the compiler may optimize this code, if it finds that the program **never** ever makes any attempt to change the value of some_int, so it may be tempted to optimize the while loop by changing it from while(some_int == 100) to *something* which is equivalent to while(true) so that the execution could be fast (since the condition in while loop appears to be true always). *(if the compiler doesn't optimize it, then it has to fetch the value of some_int and compare it with 100, in each iteration which obviously is a little bit slow.)*
However, sometimes, optimization (of some parts of your program) may be **undesirable**, because it may be that someone else is changing the value of some_int from **outside the program which compiler is not aware of**, since it can't see it; but it's how you've designed it. In that case, compiler's optimization would **not** produce the desired result!

18. Operator overriding is not possible in C++.