Garbage Collection & Dangling Reference

```
int main(){
        int *p = new int;
        *p = 3
        cout << *p << endl; // prints 3
        delete p;
        cout << *p << endl; // prints random numbers
        float *q = new float;
        *q = 5; // may overwrite space where p points
        cout << *p << endl; // undefined behavior because p has been reclaimed

}
```

Reference counts

```
For every data in heap you remember a reference (how many pointer point to that
data). If it reaches 0, you know it is garbage.
```

Mark and sweep

```
Considers all objects in the heap as potentially reachable. Then everything
directly accessible by the stack is marked. Everything unmarked is collected by
garbage collector and frees up memory for future allocation.
```

Stop and copy

```
Divides the available memory into two semispaces: the "from-space" and the "to-
space." The basic idea is to move live objects from the from-space to the to-space,
leaving behind only the live objects in the to-space. Reduces fragmentation.

Fragmentation -  where memory becomes divided into small, non-contiguous blocks,
making it challenging to allocate a contiguous block of memory for a new object or
data structure.
```

Generational

```
automatic memory management that takes advantage of the observation that most
objects have a short lifespan. It divides the heap into multiple generations based
on the age of objects, and it applies different garbage collection strategies to
each generation. The two main generations are the young generation and the old
generation.
```