



#ASLI ENGINEERING

Why languages have Automatic Garbage Collection?



BY

ARPIT BHAYANI

Why programming languages have Automatic Garbage Collection?

Memory management:

Our programs need memory to allocate objects and access them to do the job

↙ ↘ ↗
game accounting chaiting

Every object we see on screen or "variable" we use are allocated somewhere in the memory

There are two possible places:

1. Stack



`int a=10;`



allocates `sizeof(int)`
on stack frame of
the program

* When the function returns
the variable loses its existence
and is kind of cleaned up

2. Heap [Dynamic memory allocation]



`int *books = malloc(10 * sizeof(book))`



allocates these many
bytes of memory on
heap

Someone has to clean
this mess.

Why we need Heap in the first place?

- we can have objects that are too big or too inefficient to be on Stack

eg: object to represent a device in a kernel or customer in your CRM app

- To have dynamically growing objects

eg: Arrays, LinkedList, Trees

- To have multiple functions using the same instance of object



- To not pass a gigantic object across function

Objects allocated in the heap are always

addressed by **Reference**

↓
pointer

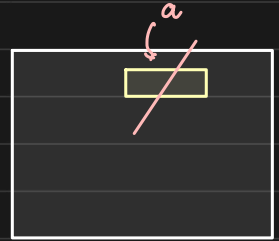
Garbage Collection : Explicit Deallocation

Programming language provide support for deallocating the allocated object on the heap

C++ : `free()` C++ : `delete()`

`free(a)`

↑
Explicit Deallocation



It is good that languages provide a way to deallocate but we cannot rely on engineers and developers to always free the memory the allocated

Because - they might forget to do so

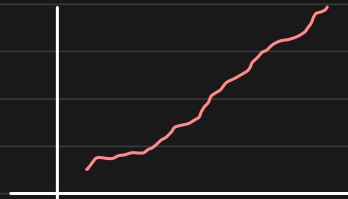
- the path in which deallocation is done is not always invoked

```
_____  
_____  
_____  
if ( _____ ) :  
    _____  
    free(a)  
    _____
```

← This path is never taken

What happens if an object is not deleted & not used ?

Memory leak

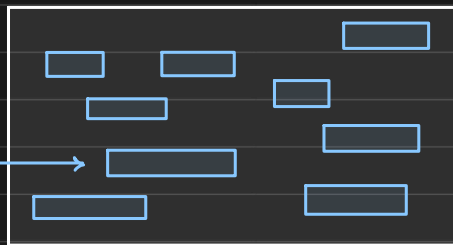


Memory Consumption
chart will steadily increase

And once this hits 100%
and process tries to allocate
a new object, the process

CRASH

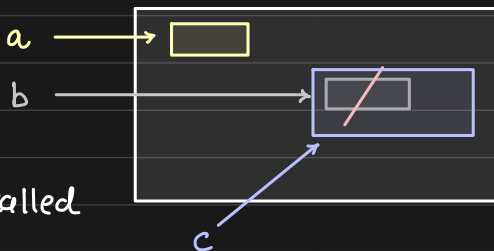
objects allocated but
not deallocated



What happens when an object is freed, but is still referenced?

Dangling Pointer

When we reference an object
that does not exist, it is called
a dangling pointer.



If we reference a dangling pointer the
results are unpredictable

The process might or might not crash
because of Dangling Pointer

Best case
↑

You would hope that
it crashes

Hence we get unpredictable behaviour

Because of these reasons, the runtime engines of
the programming languages provide a way
to do Automatic Garbage Collection



More reliable



Reduces Human
Efforts



Not prone to
human errors