**Memory Leak**

**What:**

A memory leak occurs when a piece (or pieces) of memory that was previously allocated by a programmer is not properly deallocated by the programmer. Even though that memory is no longer in use by the program, it is still “reserved”, and that piece of memory can not be used by the program until it is properly deallocated by the programmer. That’s why it’s called a memory *leak* – because it’s like a leaky faucet in which water is being wasted, only in this case it’s computer memory.

**Why to avoid memory leak:**

It leaves chunk(s) of memory unavailable for use by the programmer. If a program has a lot of memory that hasn’t been deallocated, then that could really slow down the performance of the program. If there’s no memory left in the program because of memory leaks, then that could of course cause the program to crash.

**#01\_Pgm**

#include<iostream>

using namespace std;

void function\_memoryLeak()

{

int \*ptr = new int [10]; // 10\*4 40 bytes of memory is dynamically allocated

//delete ptr;

}

int main()

{

function\_memoryLeak();

return 0;

}

**How to avoid memory leak:**

Instead of managing memory manually, try to use smart pointers where applicable.

* use std::string instead of char \*. The std::string class handles all memory management internally, and it’s fast and well-optimized.
* Never use a raw pointer unless it’s to interface with an older lib.
* The best way to avoid memory leaks in C++ is to have as few new/delete calls at the program level as possible – ideally NONE. Anything that requires dynamic memory should be buried inside an RAII object that releases the memory when it goes out of scope. RAAI allocate memory in constructor and release it in destructor, so that memory is garanteed to be deallocated when the variable leave the current scope.
* Allocate memory by new keyword and deallocate memory by delete keyword and write all code between them.