Memory leaks refer to a type of software bug that occurs when a program fails to release memory that is no longer needed, causing the program to consume more memory than necessary. Over time, the accumulation of these unreleased memory blocks can result in a significant decrease in system performance, eventually leading to crashes or other malfunctions. Memory leaks can occur in a variety of programming languages and environments, including C, C++, Java, and JavaScript. They can be caused by a variety of factors, including incorrect use of memory allocation functions, failure to free dynamically allocated memory, and circular references in object-oriented programming.

Memory leaks in C happen for **three core reasons**:

- we do not free the memory that is no longer needed
- we do try to free the memory but we do not have the reference to it (dangling pointer)
- we try to free the memory using the wrong function

Types of **Memory Leaks** or Common Dynamic Memory Problems in C are:

- Dynamic memory allocation **malloc** and not deallocating it.
- Dynamic memory allocation **malloc** and deallocate it with delete.
- Dynamic memory allocation with **new** and not deallocating with delete.
- Dynamic memory allocation with **new[ ]** and deallocating with delete.
- Dynamic memory allocation **new** and deallocate it with free.

As we all know that **malloc** is used to allocate dynamic memory in C,and we need to free this memory to avoid memory leak but sometimes instead of using **free()**
we use **delete** keyword to free this memory which is used in C++ to free Dynamic memory.

To free up Dynamic memory allocated to an array we cannot directly use **delete** operator.

Use delete[] to free dynamic memory allocated to an array.

Example 1

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
   char *data1 = "Amrita";
   char *data2 = "TIFAC";
   printf("data1=%p\t%s\n",(void*)data1,data1);
   printf("data2=%p\t%s\n",(void*)data2,data2);
   data2 = data1; // both data1 and data2 point to the same memory
```

```
    printf("data1=%p\t%s\n",(void*)data1,data1);
    printf("data2=%p\t%s\n",(void*)data2,data2);
    free(data1);
    free(data2); // does not free original data2
    return 0;
}
```

Example 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    //Initialize the seed of the random number generator
    srand(time(NULL));

    int i = 0;

    while(i++ < 5)
    {
        int* ptr = malloc(sizeof(int));
        *ptr = rand() % 100;
        printf("Generated random number %d\n", *ptr);
    }
}
```

Solution

```c
while(i++ < 5)
{
    int* ptr = malloc(sizeof(int));
    *ptr = rand() % 100;
    printf("Generated random number %d\n", *ptr);
    free(ptr);
}
```