RV Educational Institutions ®
RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya
Technological University,
Belagavi

Approved by AICTE,
New Delhi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS - CS235AI

REPORT

Submitted by

Rohit Suresh	1RV22CS163
Pramath KP	1RV22CS142

Computer Science and Engineering 2023-2

Advanced Methodologies for Memory Leak Detection in Software Programs

Problem Statement:

Create a memory leak detection tool to identify and report unallocated memory instances in software. The tool should analyze runtime memory usage, pinpoint source code locations of leaks, and provide a detailed report. It must be efficient, cross-platform, and support various programming languages. Additionally, include comparative documentation among various tools and examples.

Introduction

Memory leak detection is a critical aspect of software development aimed at identifying and resolving memory leaks within computer programs. A memory leak occurs when a program fails to release memory that it has allocated but no longer needs, leading to a gradual depletion of available memory resources. Over time, memory leaks can result in performance degradation, instability, and even system crashes.

Memory leak detection involves tools and techniques designed to identify these leaks, allowing developers to address them before they become significant issues.

By incorporating memory leak detection into the software development lifecycle, developers can identify and address memory leaks early, ensuring that their applications remain stable, efficient, and resilient over time.

Comparative Study:

VALGRIND	GDB
It is a memory leak detection tool used to check if there are any untoward memory allocations, or leaks in the program	It is a command line debugger which helps us to decode where the error might be occurring in the program
Valgrind uses a plugin architecture which helps users to add custom plug-ins and can help in detecting very specific memory leaks	Breakpoints and Stepping: in the GDB command line we can set multiple breakpoints in our code, and step through each line of the code, making it user-friendly.
Valgrind is portable across all operating systems, like Linux, MacOS etc. making it very versatile and stable.	During debugging, the developers can run blocks of the code dynamically, making it easier for finding out errors.
Though primarily used for C and C++, valgrind, through its extensions can be used to debug fortran and ada as well	GDB supports multithreaded programming, allowing users to access and debug multiple threads at the same time.

ELECTRIC FENCE	ABRT	SAR
Detection Capability:Limited to buffer overflows.	Detection Capability: Detects various memory errors, including leaks.	Detection Capability:directly detects leaks by monitoring memory usage trends.
Moderate to high overhead due to page allocation.	low to moderate overhead, with slight binary size and runtime performance overhead.	Minimal overhead as it collects system activity data passively.
Relatively simple.to use	Requires compiler instrumentation and integration.	Requires interpretation and correlation with other tools.

Focus on detecting memory errors within the application.	: Focus on detecting memory errors within the application.	Monitors system-wide activity, providing a broader context.

LEAK SANITIZER	MTRACE	MEMLEAX
Instruments memory allocation functions to track leaked memory.	Monitors memory allocation and deallocation calls, providing detailed reports on memory usage.	Tracks memory allocations and deallocations to detect leaks.
Low to moderate overhead.	Generally lightweight, with minimal impact on performance.	Lightweight with minimal runtime overhead.
Integrated into compilers like GCC and Clang, making it easy to enable during compilation without additional setup.	Requires linking with libmcheck and setting environment variables, but relatively simple to use.	Straightforward integration into development workflows.
Works on both unix and windows based systems	Works majorly on unix systems	Compatible on all systems and environments

Relevant Operating system Concept and API/System calls used:

A memory leak occurs when programmers create a memory in a heap and forget to delete it

The consequence of the memory leak is that it reduces the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated, all or part of the system or device stops working correctly, the application fails, or the system slows down vastly.

Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate.

In this code, the following API/system calls are used:

- malloc: Allocates memory dynamically.
- calloc: Allocates memory for an array of elements, initializing them to zero.
- free: Deallocates previously allocated memory.
- strcpy: Copies a string.
- memset: Fills a block of memory with a particular value.
- strncpy: Copies a certain amount of characters from one string to another.
- sprintf: Prints formatted output to a string.
- fopen: Opens a file.
- fwrite: Writes data to a file.
- fclose: Closes a file.

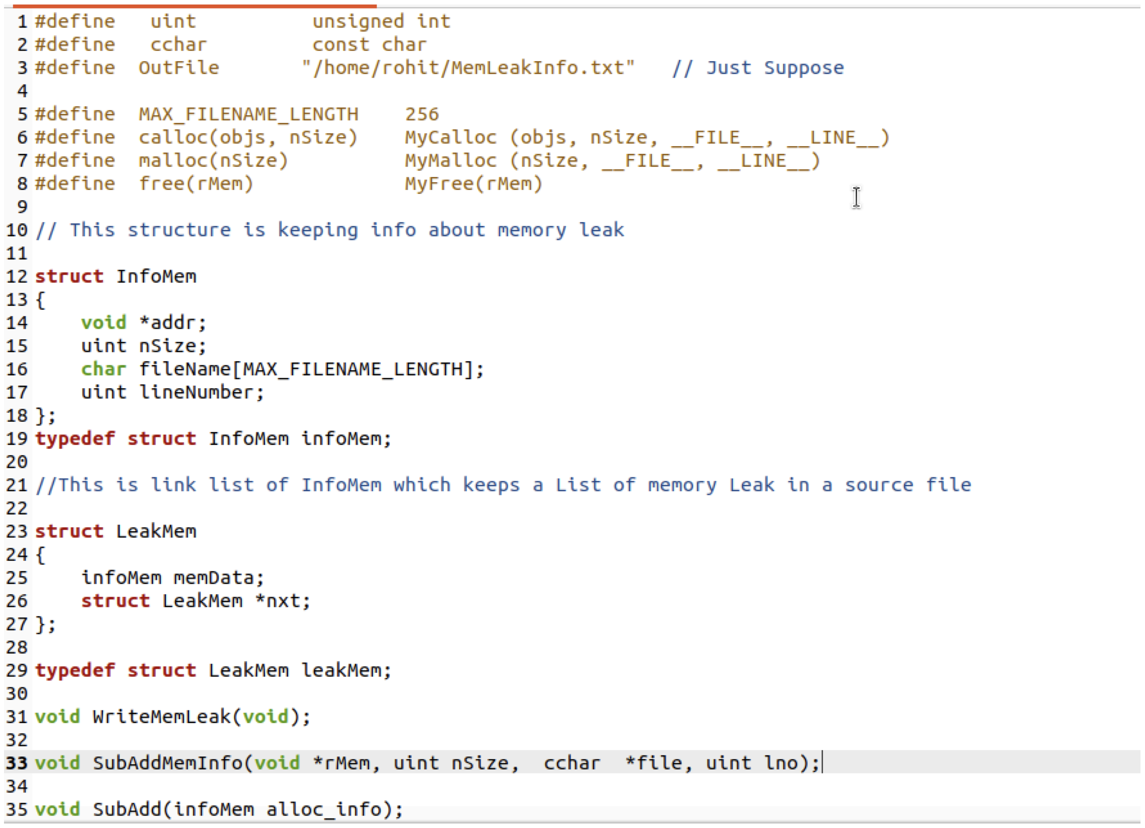
Design of the problem:

This code is a memory leak detection mechanism implemented in C. It overrides the default `malloc`, `calloc`, and `free` functions to keep track of memory allocations and deallocations. Here's a brief explanation of the code:

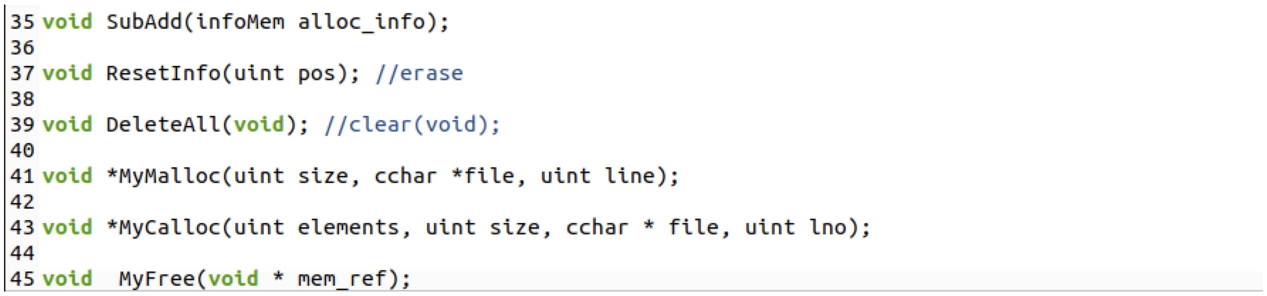
1. The code defines structures and functions to track memory allocations and deallocations.
2. It provides functions like `MyMalloc`, `MyCalloc`, `MyFree` to replace the standard memory allocation functions and add information about memory allocations to a linked list.
3. The `SubAdd` function adds memory allocation information to the linked list.
4. The `ResetInfo` function removes memory allocation information from the linked list based on the position.
5. The `DeleteAll` function deletes all elements from the linked list.
6. The `WriteMemLeak` function writes information about memory leaks to a file, including the file name, memory address, size of memory leak, and line number where deallocation did not occur.

Source Code:

findeak.h
header file

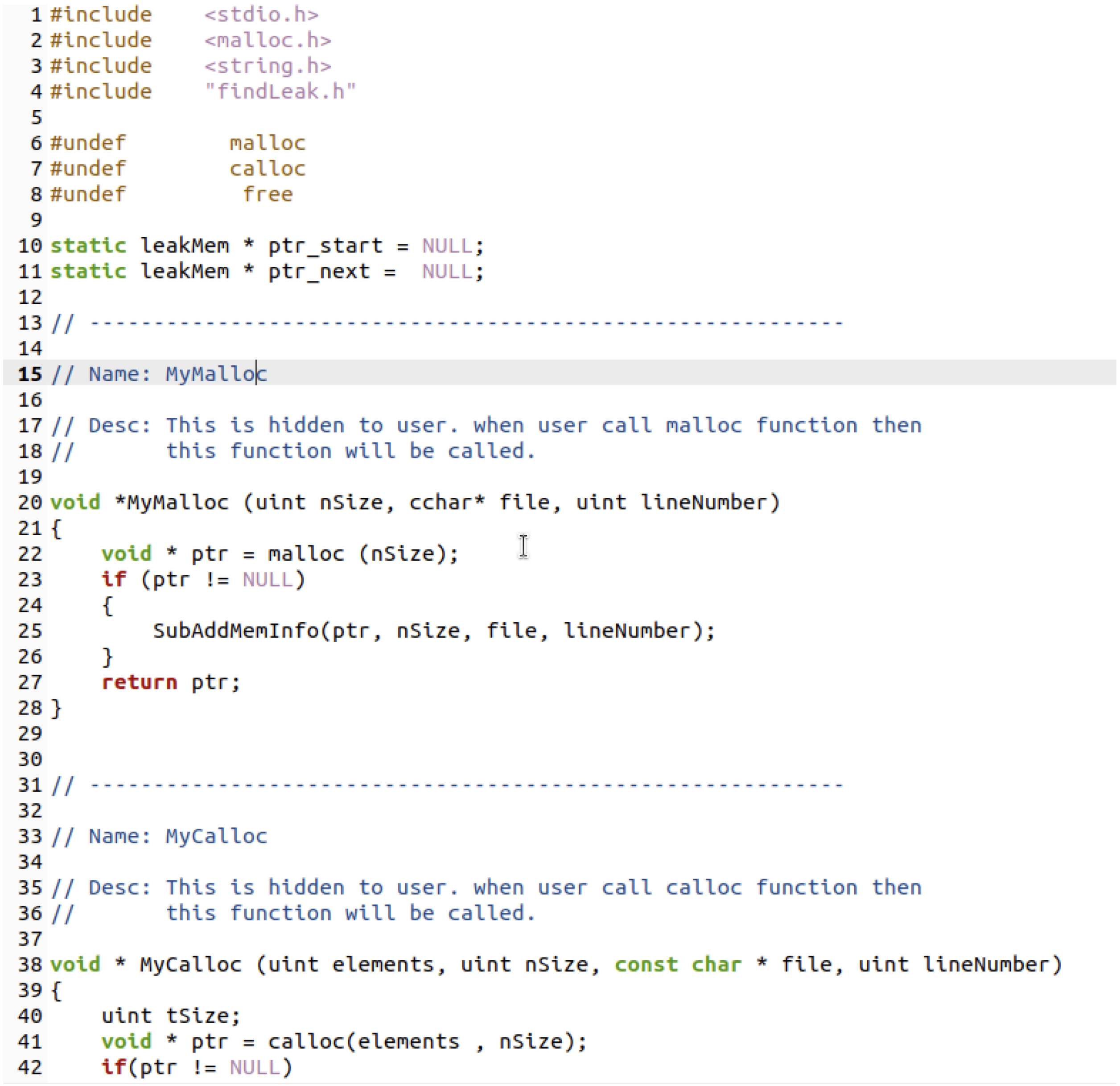


```c
#define   uint          unsigned int
#define   cchar         const char
#define  OutFile       "/home/rohit/MemLeakInfo.txt"   // Just Suppose

#define  MAX_FILENAME_LENGTH    256
#define  calloc(objs, nSize)    MyCalloc (objs, nSize, __FILE__, __LINE__)
#define  malloc(nSize)          MyMalloc (nSize, __FILE__, __LINE__)
#define  free(rMem)             MyFree(rMem)

// This structure is keeping info about memory leak

struct InfoMem
{
    void *addr;
    uint nSize;
    char fileName[MAX_FILENAME_LENGTH];
    uint lineNumber;
};
typedef struct InfoMem infoMem;

//This is link list of InfoMem which keeps a List of memory Leak in a source file

struct LeakMem
{
    infoMem memData;
    struct LeakMem *nxt;
};

typedef struct LeakMem leakMem;

void WriteMemLeak(void);

void SubAddMemInfo(void *rMem, uint nSize,  cchar  *file, uint lno);

void SubAdd(infoMem alloc_info);
```

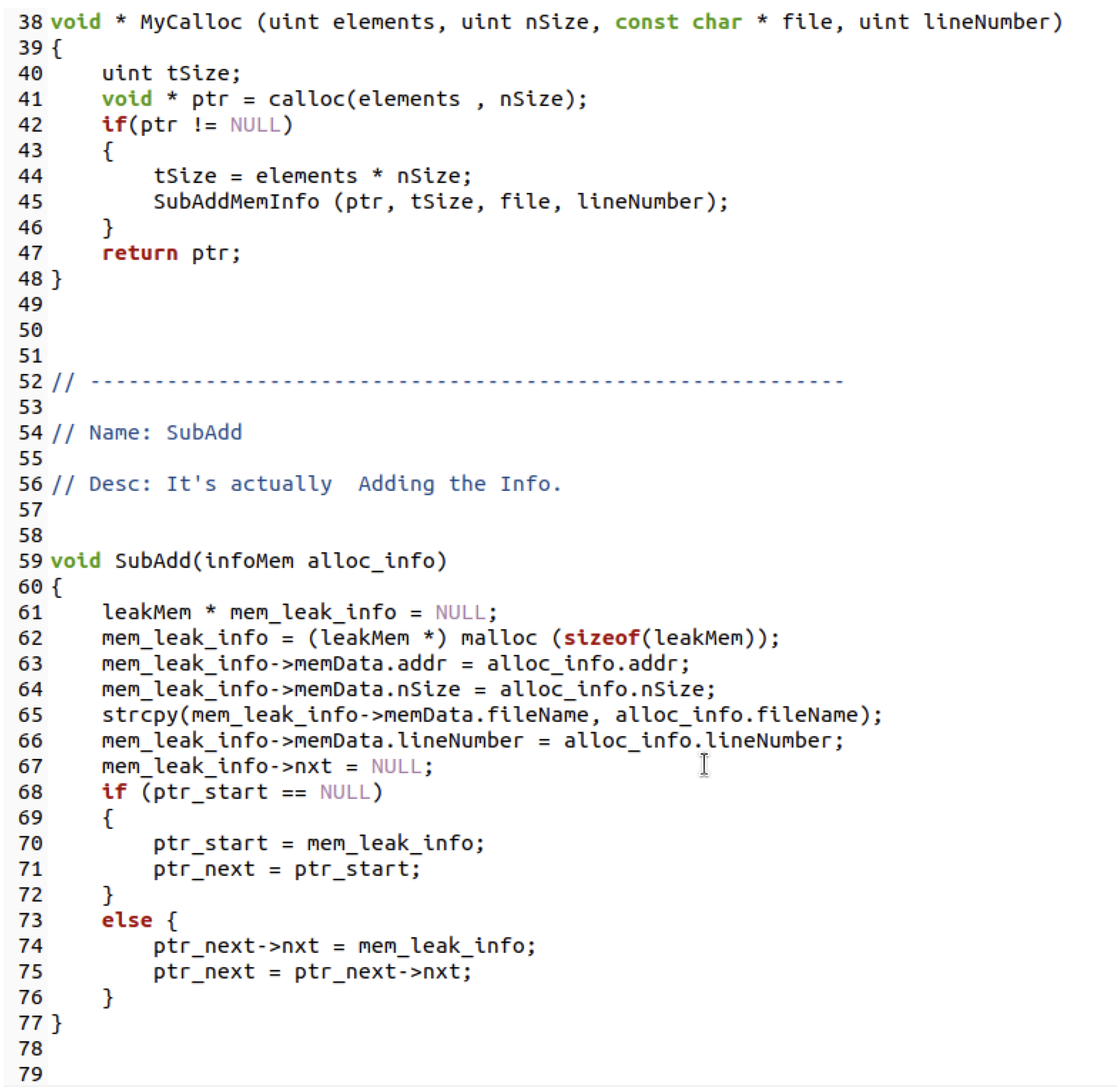


```c
void SubAdd(infoMem alloc_info);

void ResetInfo(uint pos); //erase

void DeleteAll(void); //clear(void);

void *MyMalloc(uint size, cchar *file, uint line);

void *MyCalloc(uint elements, uint size, cchar * file, uint lno);

void  MyFree(void * mem_ref);
```

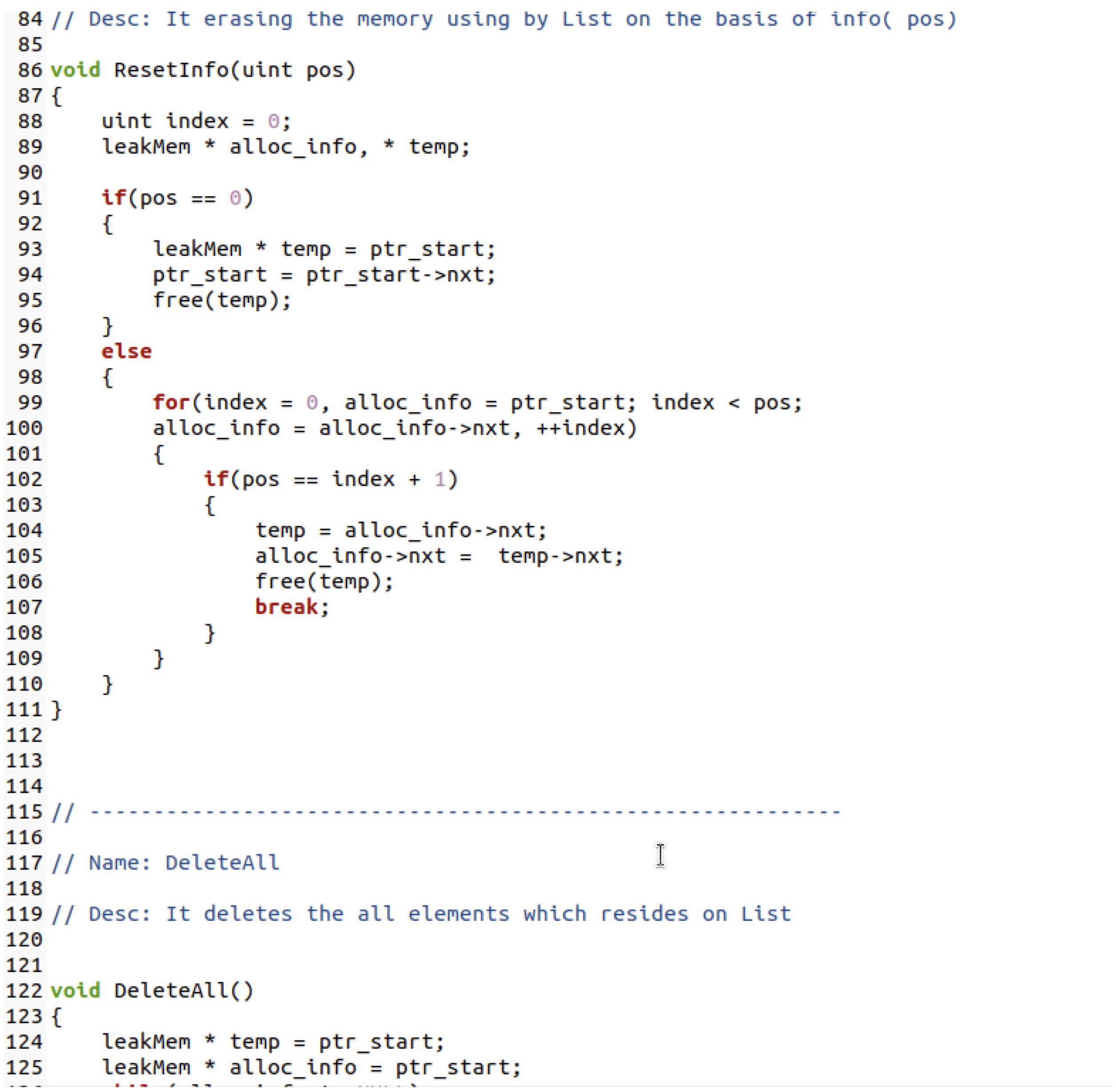
FindLeak.c



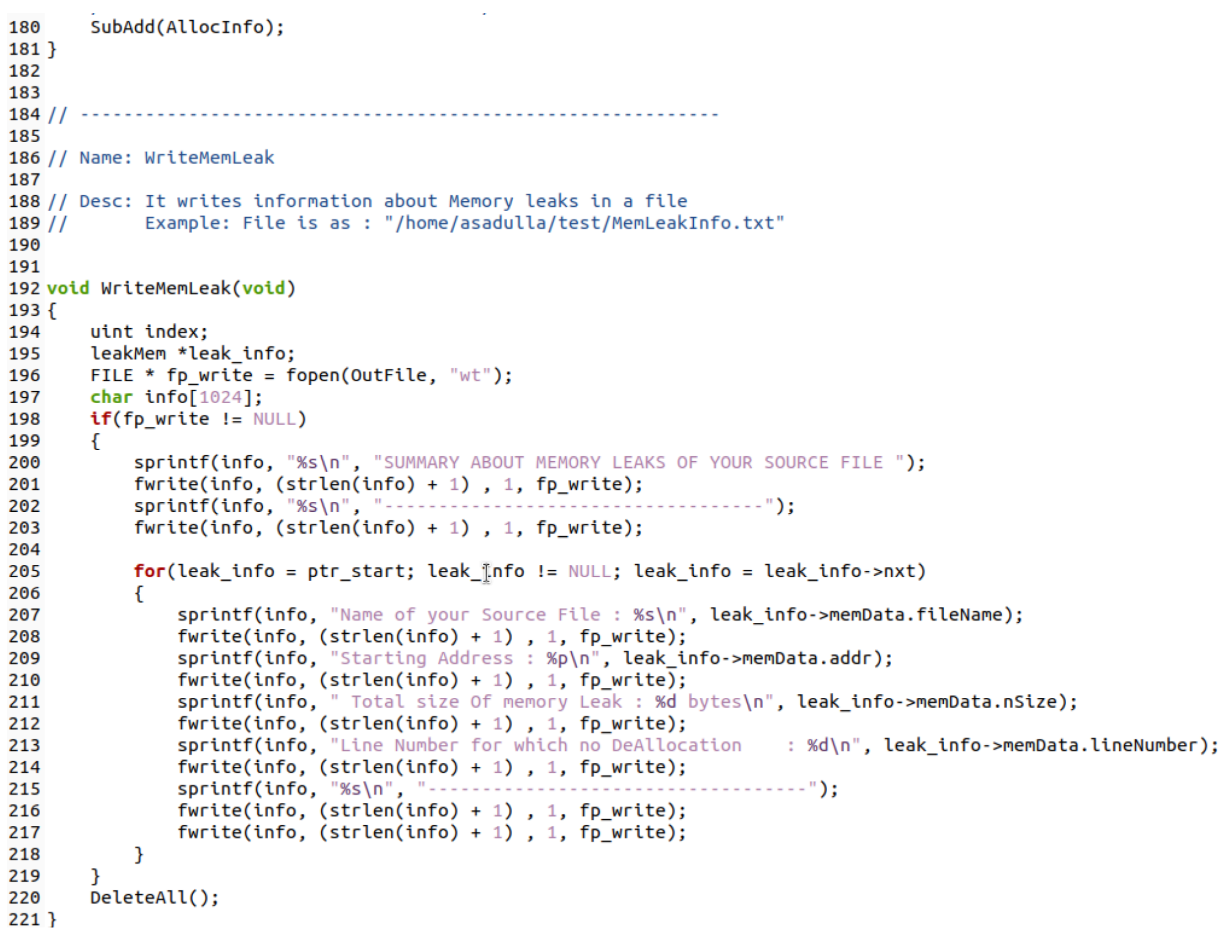
```c
#include    <stdio.h>
#include    <malloc.h>
#include    <string.h>
#include    "findLeak.h"

#undef        malloc
#undef        calloc
#undef         free

static leakMem * ptr_start = NULL;
static leakMem * ptr_next =  NULL;

// -----------------------------------------------------------

// Name: MyMalloc

// Desc: This is hidden to user. when user call malloc function then
//       this function will be called.

void *MyMalloc (uint nSize, cchar* file, uint lineNumber)
{
    void * ptr = malloc (nSize);
    if (ptr != NULL)
    {
        SubAddMemInfo(ptr, nSize, file, lineNumber);
    }
    return ptr;
}


// -----------------------------------------------------------

// Name: MyCalloc

// Desc: This is hidden to user. when user call calloc function then
//       this function will be called.

void * MyCalloc (uint elements, uint nSize, const char * file, uint lineNumber)
{
    uint tSize;
    void * ptr = calloc(elements , nSize);
    if(ptr != NULL)
```



```c
void * MyCalloc (uint elements, uint nSize, const char * file, uint lineNumber)
{
    uint tSize;
    void * ptr = calloc(elements , nSize);
    if(ptr != NULL)
    {
        tSize = elements * nSize;
        SubAddMemInfo (ptr, tSize, file, lineNumber);
    }
    return ptr;
}



// -----------------------------------------------------------

// Name: SubAdd

// Desc: It's actually  Adding the Info.


void SubAdd(infoMem alloc_info)
{
    leakMem * mem_leak_info = NULL;
    mem_leak_info = (leakMem *) malloc (sizeof(leakMem));
    mem_leak_info->memData.addr = alloc_info.addr;
    mem_leak_info->memData.nSize = alloc_info.nSize;
    strcpy(mem_leak_info->memData.fileName, alloc_info.fileName);
    mem_leak_info->memData.lineNumber = alloc_info.lineNumber;
    mem_leak_info->nxt = NULL;
    if (ptr_start == NULL)
    {
        ptr_start = mem_leak_info;
        ptr_next = ptr_start;
    }
    else {
        ptr_next->nxt = mem_leak_info;
        ptr_next = ptr_next->nxt;
    }
}


```

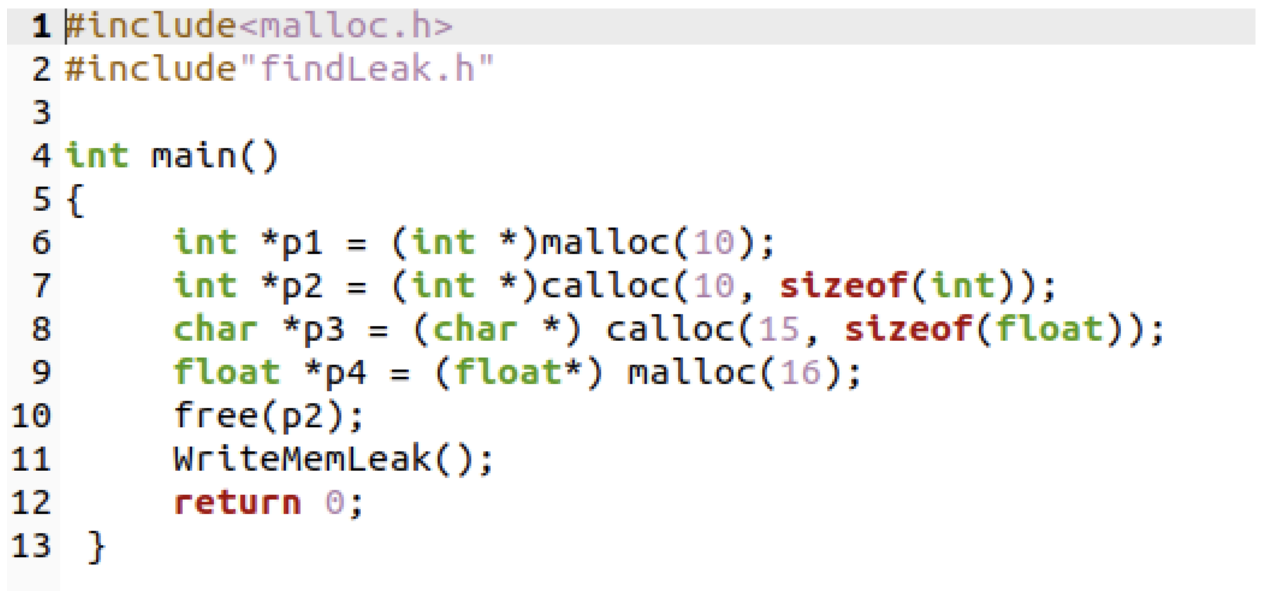


```c
// Desc: It erasing the memory using by List on the basis of info( pos)

void ResetInfo(uint pos)
{
    uint index = 0;
    leakMem * alloc_info, * temp;

    if(pos == 0)
    {
        leakMem * temp = ptr_start;
        ptr_start = ptr_start->nxt;
        free(temp);
    }
    else
    {
        for(index = 0, alloc_info = ptr_start; index < pos;
        alloc_info = alloc_info->nxt, ++index)
        {
            if(pos == index + 1)
            {
                temp = alloc_info->nxt;
                alloc_info->nxt =  temp->nxt;
                free(temp);
                break;
            }
        }
    }
}



// -----------------------------------------------------------

// Name: DeleteAll

// Desc: It deletes the all elements which resides on List


void DeleteAll()
{
    leakMem * temp = ptr_start;
    leakMem * alloc_info = ptr_start;
```



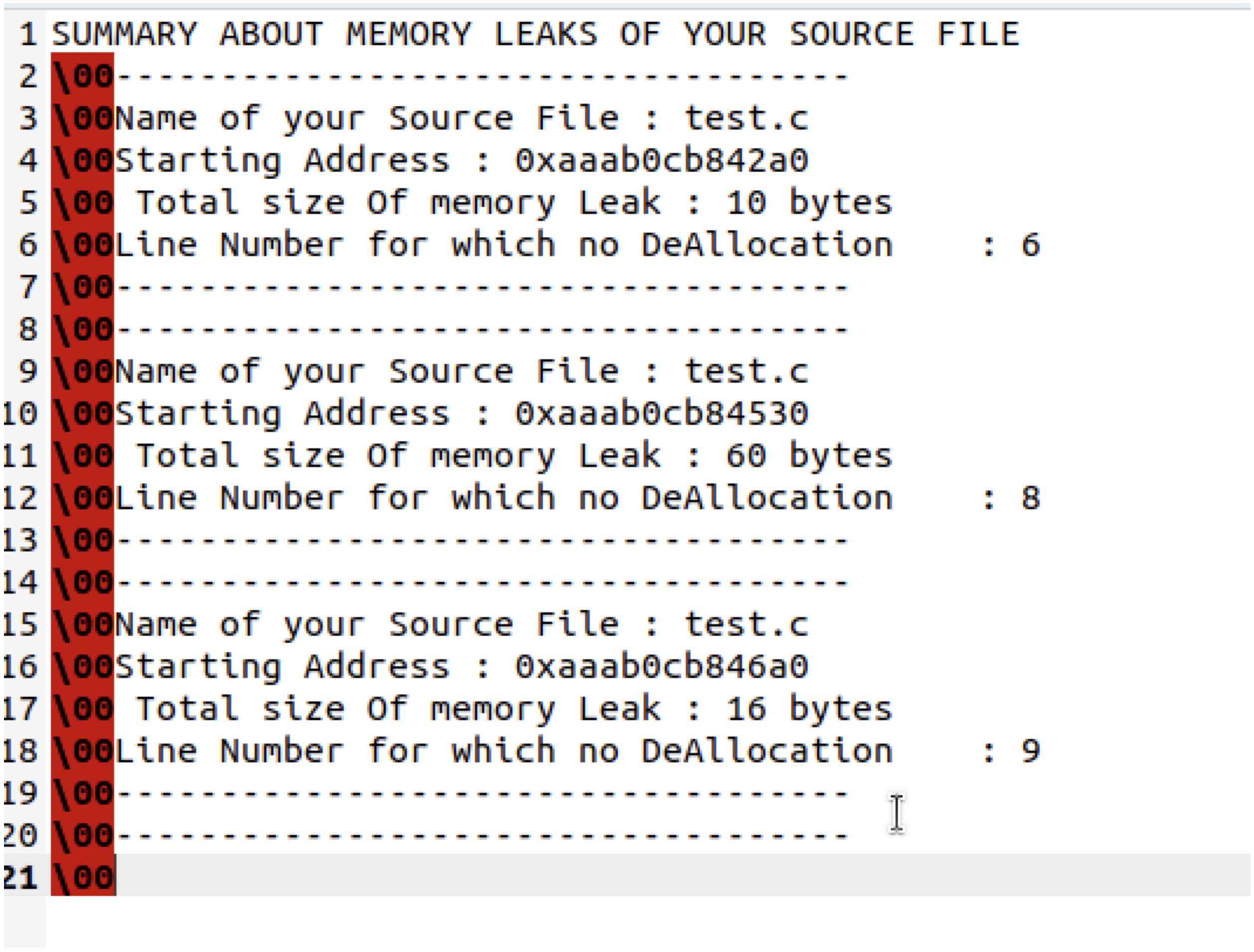
```c
    SubAdd(AllocInfo);
}


// -----------------------------------------------------------

// Name: WriteMemLeak

// Desc: It writes information about Memory leaks in a file
//       Example: File is as : "/home/asadulla/test/MemLeakInfo.txt"


void WriteMemLeak(void)
{
    uint index;
    leakMem *leak_info;
    FILE * fp_write = fopen(OutFile, "wt");
    char info[1024];
    if(fp_write != NULL)
    {
        sprintf(info, "%s\n", "SUMMARY ABOUT MEMORY LEAKS OF YOUR SOURCE FILE ");
        fwrite(info, (strlen(info) + 1) , 1, fp_write);
        sprintf(info, "%s\n", "-----------------------------------");
        fwrite(info, (strlen(info) + 1) , 1, fp_write);

        for(leak_info = ptr_start; leak_info != NULL; leak_info = leak_info->nxt)
        {
            sprintf(info, "Name of your Source File : %s\n", leak_info->memData.fileName);
            fwrite(info, (strlen(info) + 1) , 1, fp_write);
            sprintf(info, "Starting Address : %p\n", leak_info->memData.addr);
            fwrite(info, (strlen(info) + 1) , 1, fp_write);
            sprintf(info, " Total size Of memory Leak : %d bytes\n", leak_info->memData.nSize);
            fwrite(info, (strlen(info) + 1) , 1, fp_write);
            sprintf(info, "Line Number for which no DeAllocation    : %d\n", leak_info->memData.lineNumber);
            fwrite(info, (strlen(info) + 1) , 1, fp_write);
            sprintf(info, "%s\n", "-----------------------------------");
            fwrite(info, (strlen(info) + 1) , 1, fp_write);
            fwrite(info, (strlen(info) + 1) , 1, fp_write);
        }
    }
    DeleteAll();
}
```

test.c



```c
#include<malloc.h>
#include"findLeak.h"

int main()
{
        int *p1 = (int *)malloc(10);
        int *p2 = (int *)calloc(10, sizeof(int));
        char *p3 = (char *) calloc(15, sizeof(float));
        float *p4 = (float*) malloc(16);
        free(p2);
        WriteMemLeak();
        return 0;
 }
```

MemLeakInfo.txt



```
SUMMARY ABOUT MEMORY LEAKS OF YOUR SOURCE FILE
\00-----------------------------------
\00Name of your Source File : test.c
\00Starting Address : 0xaaab0cb842a0
\00 Total size Of memory Leak : 10 bytes
\00Line Number for which no DeAllocation    : 6
\00-----------------------------------
\00-----------------------------------
\00Name of your Source File : test.c
\00Starting Address : 0xaaab0cb84530
\00 Total size Of memory Leak : 60 bytes
\00Line Number for which no DeAllocation    : 8
\00-----------------------------------
\00-----------------------------------
\00Name of your Source File : test.c
\00Starting Address : 0xaaab0cb846a0
\00 Total size Of memory Leak : 16 bytes
\00Line Number for which no DeAllocation    : 9
\00-----------------------------------
\00-----------------------------------
\00
```

Conclusion:

Through this project we’ve realized the research done on this topic was lacking and hence it becomes an absolute necessity for us to continue research on this. We have learnt the usage of multiple tools, their inner workings and how it can make the life of a software developer easier, and help him to build robust, useful and memory efficient applications. A comparative study would be helpful for a person to choose the best suited tool for his requirement.