RV Educational Institutions ®
RV College of Engineering ®

Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi

Approved by AICTE, New Delhi, Accredited By NAAC, Bengaluru And NBA, New Delhi

**DEPARTMENT OF**

**COMPUTER SCIENCE AND ENGINEERING**

**OS EL REPORT**

# EL TOPIC : MEMORY CONSUMPTION PROFILING AND COMPRESSION SYSTEM

**Submitted by**

**Nayana Prakash Naik- 1RV22CS126**

**Nishchint Tiku -1RV22CS131**

**Pratham Chib-1RV22CS146**

**Submitted to**

**Prof. Vishalakshi Prabhu**

## TABLE OF CONTENTS

1. INTRODUCTION
2. OBJECTIVES
3. PROBLEM SURVEY
4. PSEUDO CODE
5. METHODOLOGY
6. LITERATURE SURVEY
7. CONTENTS
8. TIMELINE
9. REFERENCES

# INTRODUCTION

WHAT IS A MEMORY CONSUMPTION PROFILER SYSTEM?

A memory leak detection system is a tool or set of tools designed to identify and report instances of memory leaks in software applications. A memory leak occurs when a program allocates memory but fails to release it properly, leading to a gradual consumption of system resources and potential degradation of performance. Memory leaks can be a significant issue in long-running applications, as they can cause the program to use more and more memory over time, eventually leading to crashes or slowdowns.

Here are some common features and approaches used in memory leak detection systems:

- **Instrumentation**: Memory leak detection tools often work by instrumenting the code, adding extra instructions to monitor memory allocations and deallocations. They may track every allocation and deallocation and analyze the patterns to detect potential leaks.
- **Heap Analysis:** Memory leak detection systems typically focus on the heap memory, where dynamic memory allocations occur. They analyze the state of the heap to identify blocks of memory that were allocated but not properly deallocated.
- **Reference Counting:** Some systems use reference counting to keep track of the number of references to each allocated memory block. If the reference count does not reach zero when the memory is expected to be released, it indicates a potential memory leak.
- **Garbage Collection:** In languages with garbage collection, memory leak detection may involve analyzing the garbage collection logs to identify areas where objects are not being collected as expected.
- **Static Analysis:** Some tools perform static analysis of the source code or compiled binaries to identify potential memory leaks  before the program is even executed.
- **Runtime Monitoring**: Memory leak detection systems often operate during the runtime of an application, monitoring memory usage and raising alerts when abnormal patterns or potential leaks are detected.
- **Integration with Development Environments**: Many memory leak detection tools integrate with popular development environments, making it easier for developers to identify and fix memory leaks during the development and testing phases

## HOW IS OUR PROJECT DIFFERENT?

We have created a website which gives every detail about memory consumption profiling and is basically divided into 3 parts.

1.**CHAT BOT**

We have created a chat-bot to answer all the queries related to memory profiling and dynamic memory allocation for a new exposed netizen in coding to understand the importance of memory management in OS

2. **PYCHARM SIMULATIONS on Dynamic Storage Allocation Solution**

The multiple partitioning method in OS is used for memory allocation to allocate blocks of memory available optimising the memory. Gives simulation on paging and virtual memory concepts- best fit, worst fit, first fit

- First fit: Allocate the first hole that is big enough.
- Best fit: Allocate the smallest hole that is big enough.
- Worst fit. Allocate the largest hole.

3. **C CODE**

Writes C commands on memory allocation-deallocation and compression

**LINKS**

- Website on Memory profiling:

  https://nishchint.my.canva.site/memory-profiler

- Python visualisation on dynamic memory allocation solution github repository:

  https://github.com/PRATHAM-CHIB/PYCHARM-SIMULATION/blob/main/opsEL.py

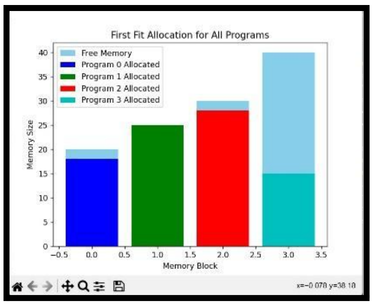
- AI Chatbot on dynamic memory allocation: https://mediafiles.botpress.cloud/aeb03cd6-a4b0-440a-8417-d2cd0308a251/webchat/bot.html

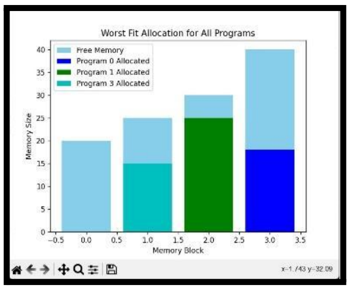
- Github repository on memory leak detection code:

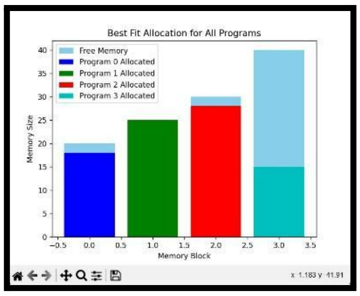
  https://github.com/PRATHAM-CHIB/memory-leak-detection

- Github repository on memory compression code:

  https://github.com/PRATHAM-CHIB/MEMORY-COMPRESSION/tree/main

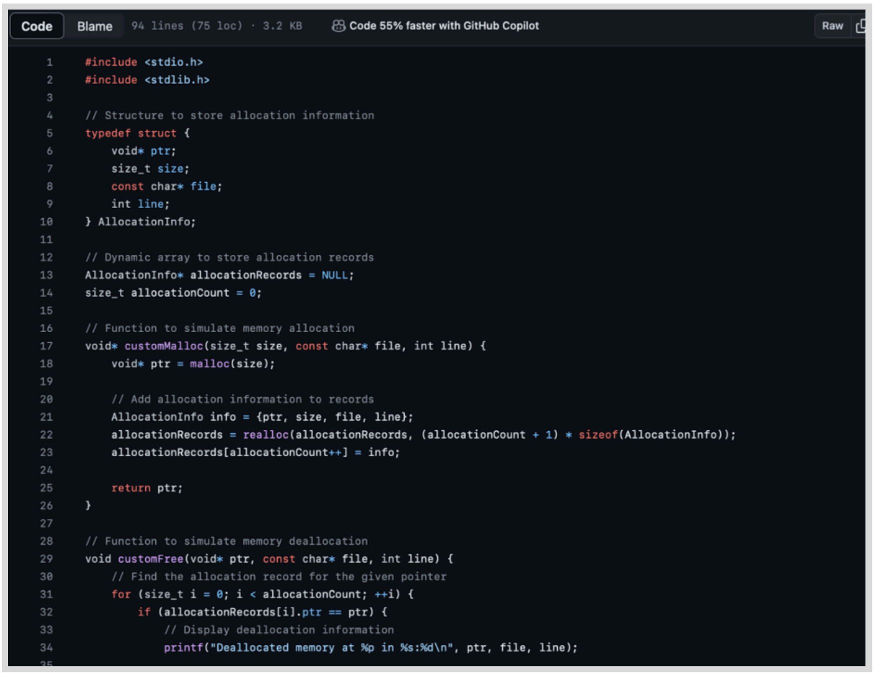




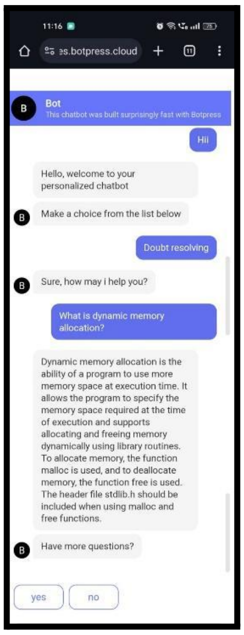


FIRST FIT WORST FIT BEST FIT

## MEMORY MANAGEMENT GITHUB CODE


Code
Blame
94 lines (75 loc) · 3.2 KB
Code 55% faster with GitHub Copilot
Raw

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

// Structure to store allocation information
typedef struct {
    void* ptr;
    size_t size;
    const char* file;
    int line;
} AllocationInfo;

// Dynamic array to store allocation records
AllocationInfo* allocationRecords = NULL;
size_t allocationCount = 0;

// Function to simulate memory allocation
void* customMalloc(size_t size, const char* file, int line) {
    void* ptr = malloc(size);

    // Add allocation information to records
    AllocationInfo info = {ptr, size, file, line};
    allocationRecords = realloc(allocationRecords, (allocationCount + 1) * sizeof(AllocationInfo));
    allocationRecords[allocationCount++] = info;

    return ptr;
}

// Function to simulate memory deallocation
void customFree(void* ptr, const char* file, int line) {
    // Find the allocation record for the given pointer
    for (size_t i = 0; i < allocationCount; ++i) {
        if (allocationRecords[i].ptr == ptr) {
            // Display deallocation information
            printf("Deallocated memory at %p in %s:%d\n", ptr, file, line);
```



Hii

Hello, welcome to your personalized chatbot

Make a choice from the list below

Doubt resolving

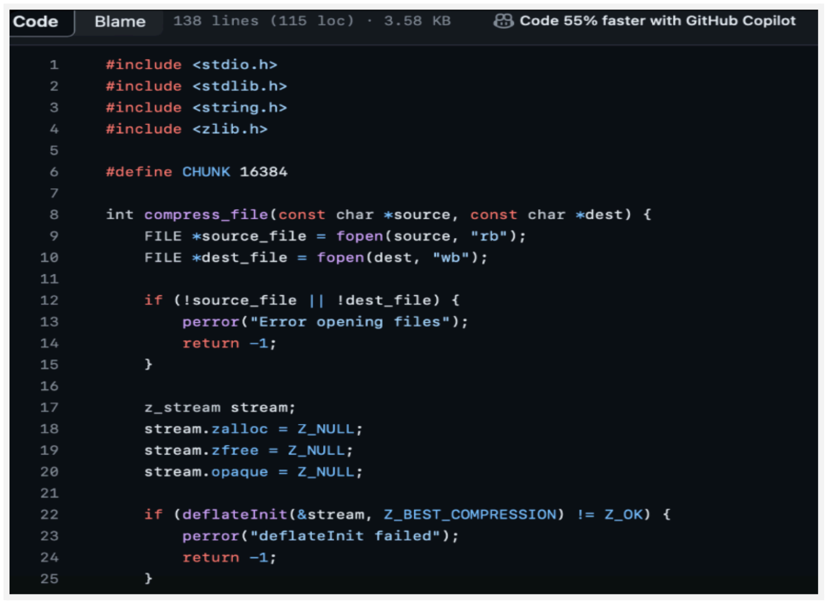
Sure, how may i help you?

What is dynamic memory allocation?

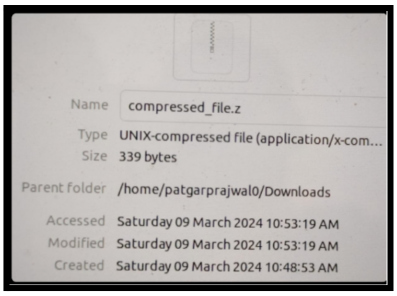
Dynamic memory allocation is the ability of a program to use more memory space at execution time. It allows the program to specify the memory space required at the time of execution and supports allocating and freeing memory dynamically using library routines. To allocate memory, the function malloc is used, and to deallocate memory, the function free is used. The header file stdlib.h should be included when using malloc and free functions.

Have more questions?

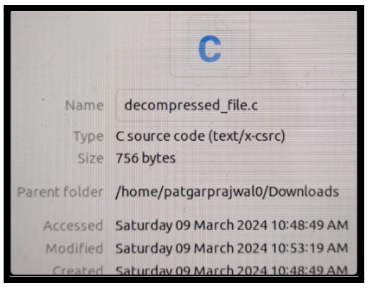
## MEMORY COMPRESSION-DEPRESSION GITHUB CODE


Code
Blame
138 lines (115 loc) · 3.58 KB
Code 55% faster with GitHub Copilot

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <zlib.h>

#define CHUNK 16384

int compress_file(const char *source, const char *dest) {
    FILE *source_file = fopen(source, "rb");
    FILE *dest_file = fopen(dest, "wb");

    if (!source_file || !dest_file) {
        perror("Error opening files");
        return -1;
    }

    z_stream stream;
    stream.zalloc = Z_NULL;
    stream.zfree = Z_NULL;
    stream.opaque = Z_NULL;

    if (deflateInit(&stream, Z_BEST_COMPRESSION) != Z_OK) {
        perror("deflateInit failed");
        return -1;
    }
```



```
Name           compressed_file.z
Type           UNIX-compressed file (application/x-com...
Size           339 bytes
Parent folder  /home/patgarprajwal0/Downloads
Accessed       Saturday 09 March 2024 10:53:19 AM
Modified       Saturday 09 March 2024 10:53:19 AM
Created        Saturday 09 March 2024 10:48:53 AM
```



```
Name           decompressed_file.c
Type           C source code (text/x-csrc)
Size           756 bytes
Parent folder  /home/patgarprajwal0/Downloads
Accessed       Saturday 09 March 2024 10:48:49 AM
Modified       Saturday 09 March 2024 10:53:19 AM
Created        Saturday 09 March 2024 10:48:49 AM
```

## Relevance of the Project to the Subject

- The project, Memory leak detection API relevance to the subject is the utilisation of various types of system calls, optimising the modern operating system, pthread library function and the implementation of various concepts of operating systems.
- We are also implementing the core backbone of our project - MEMORY MANAGEMENT UNIT (MMU), contiguous memory allocation, concepts of paging, dynamic memory allocation-freeing memory , allocating memory and memory compression techniques to highly optimise the OS.

Memory leak detection API calls in operating systems are highly relevant for several reasons:

- **Resource Management:** Memory leaks occur when a program allocates memory but fails to deallocate it properly. Over time, this can lead to the exhaustion of available memory, causing system instability and potential crashes.
- **Performance Optimization:** Memory leaks can degrade system performance by consuming available memory unnecessarily. By detecting and fixing memory leaks, developers can optimize their applications, leading to better performance and responsiveness for users.
- **Cost Reduction:** In cloud computing environments or any situation where resources are metered or scaled dynamically, memory leaks can result in increased operational costs due to the unnecessary consumption of resources.
- **Security Implications**: Memory leaks can also present security risks, as they may expose sensitive information stored in memory to unauthorized access. By promptly detecting and fixing memory leaks, developers can mitigate these security vulnerabilities and protect user data.

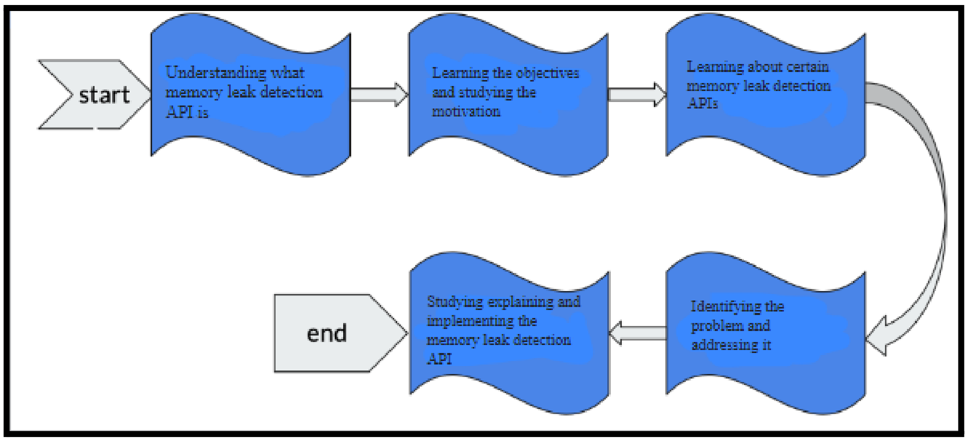
## PROBLEM SURVEY

- Improper memory management can lead to memory leaks or other issues. In the realm of software development, the occurrence of memory leaks remains a pervasive and challenging issue. Memory leaks can lead to inefficient resource utilization, degraded system performance, and ultimately, application failures. As software systems become increasingly complex and larger in scale, the identification and mitigation of memory leaks become critical for ensuring robust and reliable software.
- The primary problem lies in the manual and time-consuming nature of current memory leak detection methods. Developers often rely on traditional debugging techniques, code reviews, and static analysis tools, but these approaches may not be sufficient, especially in large and intricate codebases. Consequently, undetected memory leaks persist, leading to system instability and increased maintenance costs.

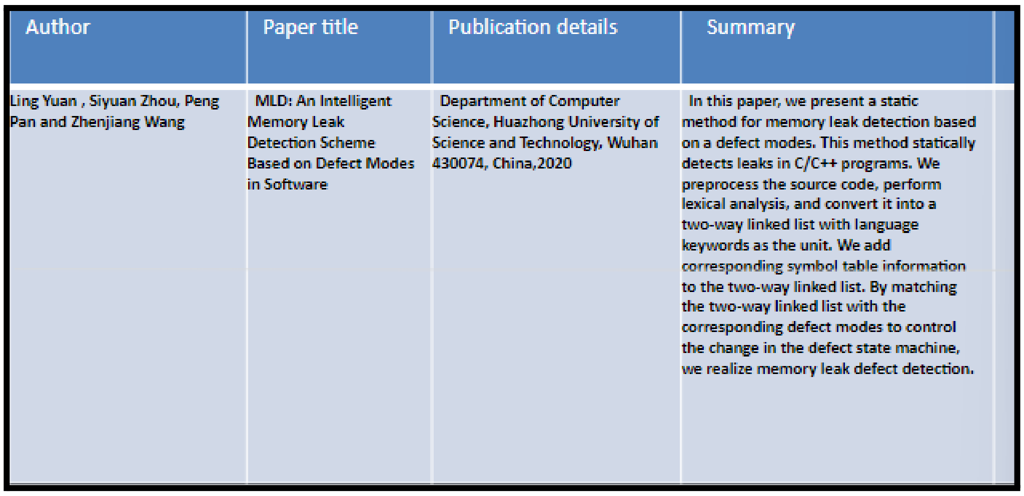
The ideal solution should:

- **Accuracy and Precision**: Provide accurate and precise identification of memory leaks to minimize false positives and negatives, ensuring that developers can trust the tool's output.
- **Integration with Development Workflow:** Seamlessly integrate with popular Integrated Development Environments (IDEs) and version control systems to enable developers to detect and resolve memory leaks as part of their regular workflow.
- R**eal-time Monitoring**: Offer real-time monitoring capabilities to detect memory leaks during runtime, allowing for proactive identification and prompt resolution
- **Scalability:** Scale efficiently to handle large and complex codebases, making it suitable for a variety of software projects ranging from small applications to enterprise-level systems
- **User-Friendly Interface:** Provide an intuitive and user-friendly interface, enabling both novice and experienced developers to navigate through the tool's features easily.
- **Minimize False Positives/Negatives**: Minimize instances of false positives and negatives to reduce the time and effort required for manual verification and correction.

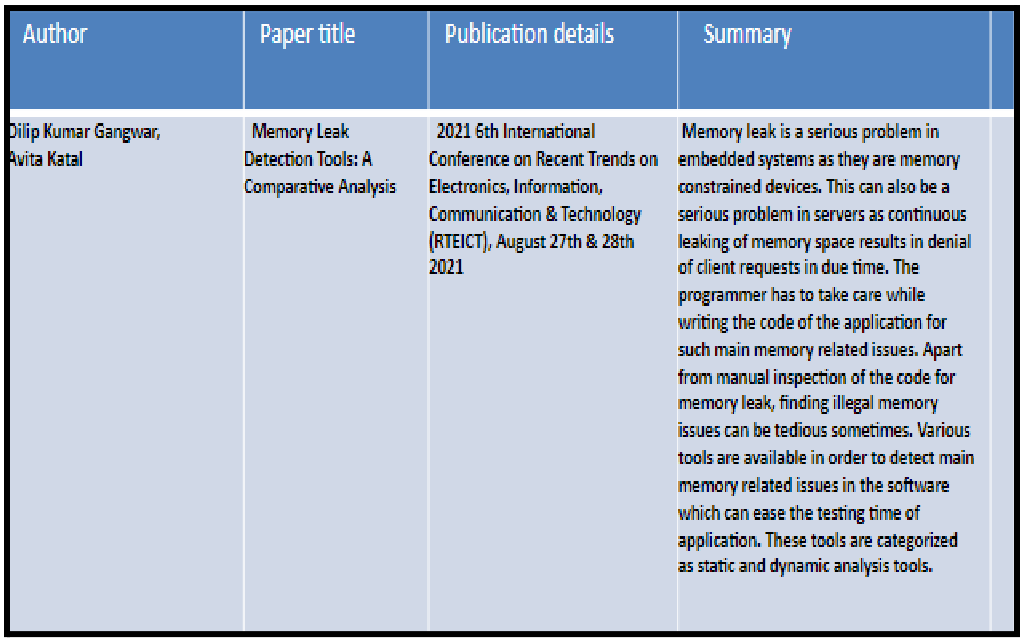
## METHODOLOGY


start
Understanding what memory leak detection API is
Learning the objectives and studying the motivation
Learning about certain memory leak detection APIs
Identifying the problem and addressing it
Studying explaining and implementing the memory leak detection API
end

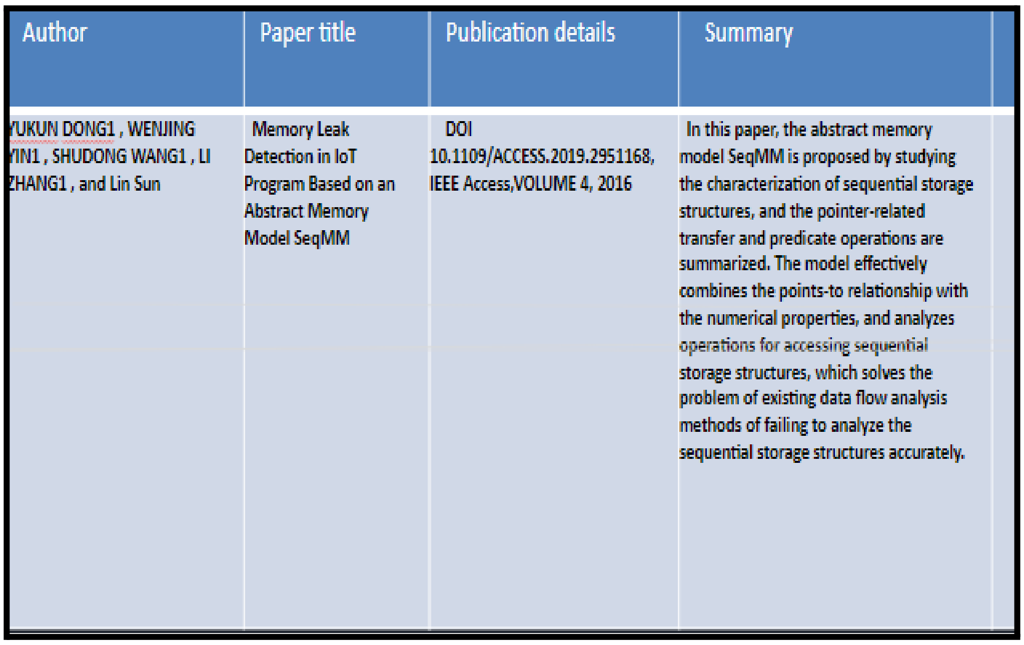
## LITERATURE SURVEY



| Author | Paper title | Publication details | Summary |
|---|---|---|---|
| Ling Yuan , Siyuan Zhou, Peng Pan and Zhenjiang Wang | MLD: An Intelligent Memory Leak Detection Scheme Based on Defect Modes in Software | Department of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China,2020 | In this paper, we present a static method for memory leak detection based on a defect modes. This method statically detects leaks in C/C++ programs. We preprocess the source code, perform lexical analysis, and convert it into a two-way linked list with language keywords as the unit. We add corresponding symbol table information to the two-way linked list. By matching the two-way linked list with the corresponding defect modes to control the change in the defect state machine, we realize memory leak defect detection. |



| Author | Paper title | Publication details | Summary |
|---|---|---|---|
| Dilip Kumar Gangwar, Avita Katal | Memory Leak Detection Tools: A Comparative Analysis | 2021 6th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), August 27th & 28th 2021 | Memory leak is a serious problem in embedded systems as they are memory constrained devices. This can also be a serious problem in servers as continuous leaking of memory space results in denial of client requests in due time. The programmer has to take care while writing the code of the application for such main memory related issues. Apart from manual inspection of the code for memory leak, finding illegal memory issues can be tedious sometimes. Various tools are available in order to detect main memory related issues in the software which can ease the testing time of application. These tools are categorized as static and dynamic analysis tools. |



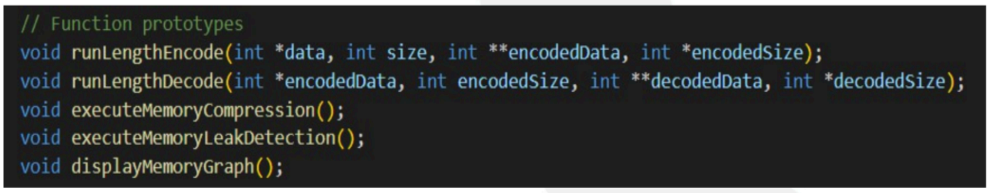
| Author | Paper title | Publication details | Summary |
|---|---|---|---|
| YUKUN DONG1 , WENJING YIN1 , SHUDONG WANG1 , LI ZHANG1 , and Lin Sun | Memory Leak Detection in IoT Program Based on an Abstract Memory Model SeqMM | DOI 10.1109/ACCESS.2019.2951168, IEEE Access,VOLUME 4, 2016 | In this paper, the abstract memory model SeqMM is proposed by studying the characterization of sequential storage structures, and the pointer-related transfer and predicate operations are summarized. The model effectively combines the points-to relationship with the numerical properties, and analyzes operations for accessing sequential storage structures, which solves the problem of existing data flow analysis methods of failing to analyze the sequential storage structures accurately. |

## OBJECTIVES

- Objectives of  memory leak detection are yielding contributions to the overall quality, reliability, and performance of software applications, leading to a more positive user experience and efficient resource utilization.
    1. **Early Identification of Issues:** Detects memory leaks early in the development process to minimize the impact on the stability and performance of the software.
    2. **Prevention of Resource Exhaustion:** Identify and rectify memory leaks to prevent the gradual accumulation of unreleased memory, which can lead to resource exhaustion and degrade the system's performance over time.
    3. **Improved System Reliability:** Enhance the reliability and robustness of software applications by eliminating memory leaks that could cause unexpected crashes, freezes, or slowdowns.
    4. **Resource Optimization:** Ensure efficient use of system resources by promptly identifying and addressing memory leaks, thereby preventing unnecessary consumption of memory.
    5. **Enhanced User Experience:** Improve the user experience by delivering software applications that are stable, responsive, and free from memory-related issues. Cost Reduction: Reduce the overall cost of software development and maintenance by addressing memory leaks early in the development life cycle. Fixing issues at later stages can be more time-consuming and expensive.
    6. **Compliance with Quality Standards:** Ensure compliance with quality standards and best practices in software development by incorporating memory leak detection as part of the testing and quality assurance processes.
    7. **Optimized System Performance:** Contribute to optimised system performance by minimizing memory leaks, which can have a   direct impact on the speed and responsiveness of an application.

# PARTIAL IMPLEMENTATION-PSEUDO CODE

## 1. Function Prototypes:



```c
// Function prototypes
void runLengthEncode(int *data, int size, int **encodedData, int *encodedSize);
void runLengthDecode(int *encodedData, int encodedSize, int **decodedData, int *decodedSize);
void executeMemoryCompression();
void executeMemoryLeakDetection();
void displayMemoryGraph();
```

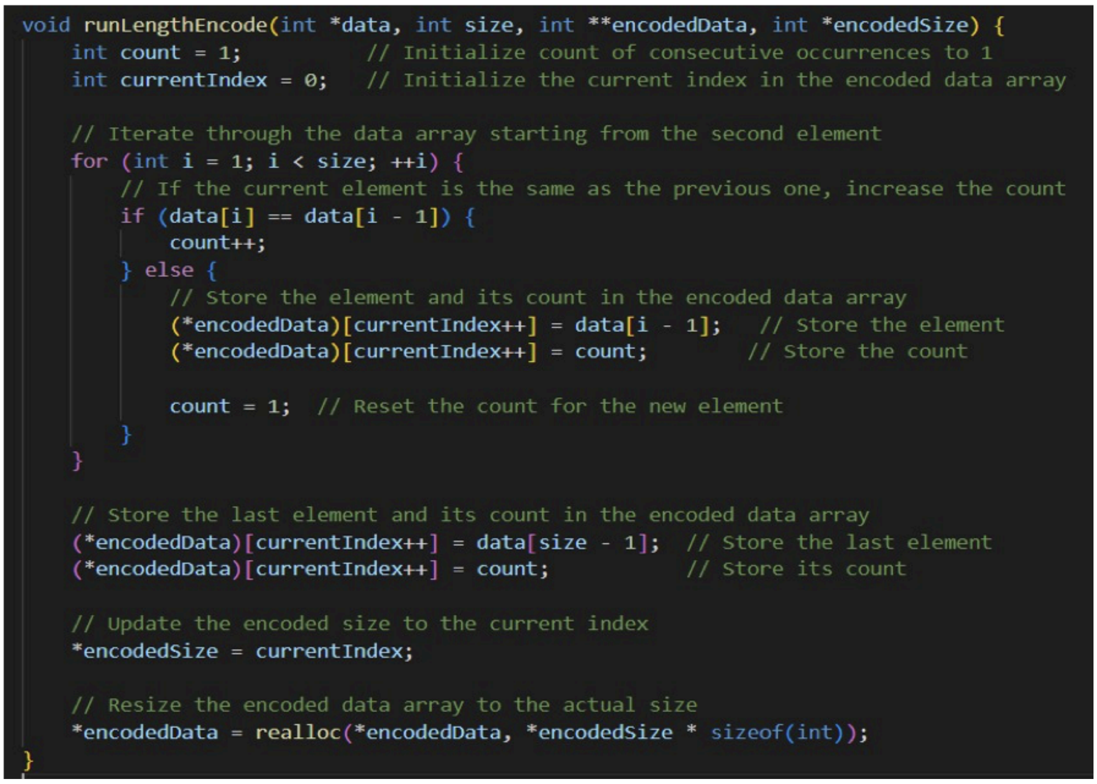
**1.runLengthEncode**: This function performs run-length encoding on an array of integers, compressing consecutive repeated elements into a single element and its count. It takes the original data, its size, and pointers to store the encoded data and its size.

**2.runLengthDecode**: This function performs the reverse operation of run-length encoding, decoding the encoded data back to its original form. It takes the encoded data, its size, and pointers to store the decoded data and its size.

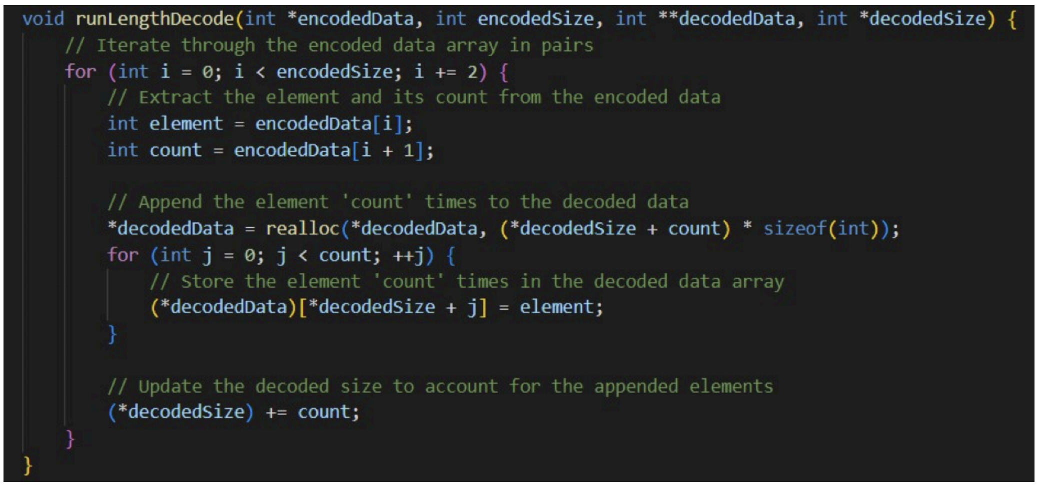
**3.executeMemoryCompression**: This function executes the memory compression process, likely involving the compression of data using run-length encoding. It is responsible for calling **runLengthEncode** to compress data.

**4.executeMemoryLeakDetection**: This function executes the memory leak detection process, likely involving checking for memory leaks in the program. It helps in identifying areas where memory has been allocated but not deallocated properly.

**5.displayMemoryGraph**: This function displays a graph of memory allocation and deallocation, providing insight into memory usage patterns during program execution. It aids in monitoring memory management and identifying potential issues such as excessive allocation or memory leaks.



```c
void runLengthEncode(int *data, int size, int **encodedData, int *encodedSize) {
    int count = 1;          // Initialize count of consecutive occurrences to 1
    int currentIndex = 0;   // Initialize the current index in the encoded data array

    // Iterate through the data array starting from the second element
    for (int i = 1; i < size; ++i) {
        // If the current element is the same as the previous one, increase the count
        if (data[i] == data[i - 1]) {
            count++;
        } else {
            // Store the element and its count in the encoded data array
            (*encodedData)[currentIndex++] = data[i - 1];   // Store the element
            (*encodedData)[currentIndex++] = count;        // Store the count

            count = 1;  // Reset the count for the new element
        }
    }

    // Store the last element and its count in the encoded data array
    (*encodedData)[currentIndex++] = data[size - 1];  // Store the last element
    (*encodedData)[currentIndex++] = count;           // Store its count

    // Update the encoded size to the current index
    *encodedSize = currentIndex;

    // Resize the encoded data array to the actual size
    *encodedData = realloc(*encodedData, *encodedSize * sizeof(int));
}
```



```c
void runLengthDecode(int *encodedData, int encodedSize, int **decodedData, int *decodedSize) {
    // Iterate through the encoded data array in pairs
    for (int i = 0; i < encodedSize; i += 2) {
        // Extract the element and its count from the encoded data
        int element = encodedData[i];
        int count = encodedData[i + 1];

        // Append the element 'count' times to the decoded data
        *decodedData = realloc(*decodedData, (*decodedSize + count) * sizeof(int));
        for (int j = 0; j < count; ++j) {
            // Store the element 'count' times in the decoded data array
            (*decodedData)[*decodedSize + j] = element;
        }

        // Update the decoded size to account for the appended elements
        (*decodedSize) += count;
    }
}
```

## TOOLS/APIs USED

1.Heap Profiling: Analyzing memory usage patterns to identify potential leaks or inefficiencies.

- Heap profiling involves analyzing the memory usage patterns of a program, particularly focusing on dynamic memory allocation (memory allocated from the heap).It helps developers understand how memory is being allocated and deallocated within their programs.
- Heap profilers often provide insights into memory allocation patterns, such as the number of allocations, the size of allocations, and the duration of memory allocations.
- These tools can help identify memory leaks (unreleased memory), excessive memory usage, and inefficient memory allocation strategies.

2. Address Sanitizer (Asan ): A tool that detects memory bugs, including leaks, in C/C++ programs

- Address Sanitizer is a dynamic memory error detector designed to find various memory safety issues such as buffer overflows, use-after-free errors, and other types of memory corruption bugs. A San instruments the code during compilation, adding runtime checks to detect invalid memory accesses. When an invalid memory access occurs (such as reading or writing beyond the bounds of an allocated memory block), A San immediately detects it and reports the error, often including information about the location in the source code where the error occurred.
- A San can significantly help in finding memory-related bugs early in the development process, making programs more robust and secure.

3. Leak Sanitizer (L San): Another tool for detecting memory leaks in C/C++ programs.

- It works by tracking memory allocations and deallocations during program execution.If memory allocated from the heap is not properly deallocated before the program exits, Leak Sanitizer identifies these leaked memory blocks and reports them.
- Leak Sanitizer helps developers identify places in the code where memory is allocated but not freed, enabling them to fix memory leaks and prevent excessive memory usage over time.

## Tools available to detect the memory leak

The tools can be categorized as

(a) Static Analysis Tools

(b) Dynamic Analysis Tools.

- Static analysis is the testing of the application without executing the application itself. Dynamic analysis tests the application by running the application.
- Both types of tools have their importance. The static analysis examines all execution paths and not only those invoked during execution.
- Dynamic analysis can find the vulnerabilities which are too complex to be found out by static analysis.

## APPLICATION OF THE PROJECT

1. Algorithm Development and Testing:
   - The run-length encoding and decoding algorithms can be used as building blocks for developing and testing other algorithms that involve data compression or encoding.
2. Application Stability:
   - Memory leaks can lead to reduced application stability over time as unreleased memory accumulates. Memory leakage detection helps maintain the stability of applications by identifying and fixing memory leaks before they impact the overall system.
3. Performance Optimization:
   - Memory leaks can contribute to performance degradation as the application consumes more memory than necessary. Memory leakage detection is essential for optimizing performance, ensuring that resources are used efficiently, and preventing unnecessary memory consumption.