1. Problem Statement

Memory management is a core aspect of systems-level programming in C. Unlike modern languages with automatic garbage collection, C requires developers to manually allocate and deallocate memory. This can often lead to memory leaks, segmentation faults, or performance issues if not handled correctly. The objective of this project is to develop a command-line tool that actively tracks memory allocations and deallocations, alerts the user to potential memory leaks, and offers a clean summary of memory activity. It acts as both a diagnostic tool and a learning aid for those working with dynamic memory in C.

3. Objectives

- To build a terminal-based memory tracking tool using C.
- To display comprehensive memory summaries, including active and freed memory.
- To help detect common issues like memory leaks and double frees.
- To reinforce understanding of pointers, dynamic memory, and structures.
- **To develop a technically relevant and meaningful project, moving away from repetitive concepts like currency converters or office/school/banking systems,text reports etc and instead focus on core programming challenges.**

4. Methodology

Application Flow:

1. **Startup:**
    - Displays a welcome message and presents a user-friendly menu.
2. **Memory Allocation:**
    - Accepts a variable name and size from the user.
    - Allocates memory using malloc() and stores all metadata (pointer, size, name).
3. **Memory Deallocation:**
    - Takes input for the variable name to be freed.
    - Validates and frees the memory, ensuring it hasn't already been deallocated.
4. **Summary Generation:**
    - Calculates and displays the total memory allocated, total freed, and remaining unfreed memory blocks.
5. **Exit Handling:**
    - On exiting the application, a final memory report is printed, warning the user of any leaks.

6. Input Details

This application does not rely on external data files. All necessary inputs are provided directly by the user during execution. Also the code focuses on using the concepts of structure as well as pointer and not relying on built-in libraries .

- Variable name (string input)
- Memory size in bytes (integer input)
- Having different functions for each case like allocation freeing memory or checking the memory occupied by current variable created as well as creating memory blocks for each variable

These inputs are stored in memory during the session and used for all track and report.

7. Code Implementation

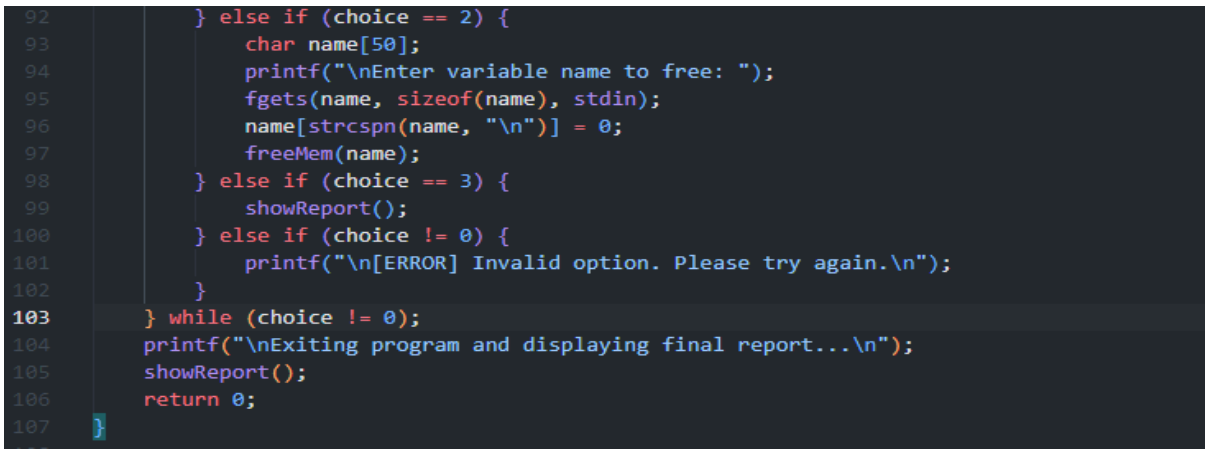
CPROJECT.c > main()

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

typedef struct {
    void* ptr;
    size_t size;
    char name[50];
    int isFree;
} Mem;
Mem memList[100];
int memCount = 0;
size_t totalGiven = 0, totalFreed = 0;

void* giveMem(size_t size, const char* name) {
    void* p = malloc(size);
    if (p) {
        totalGiven += size;
        printf("\n[INFO] Allocated %zu bytes for '%s' at address %p\n", size, name, p);
        memList[memCount].ptr = p;
        memList[memCount].size = size;
        strcpy(memList[memCount].name, name);
        memList[memCount].isFree = 0;
        memCount++;
    } else {
        printf("\n[ERROR] Allocation failed for '%s'\n", name);
    }
    return p;
}

void freeMem(const char* name) {
    for (int i = 0; i < memCount; i++) {
        if (!memList[i].isFree && strcmp(memList[i].name, name) == 0) {
            free(memList[i].ptr);
            totalFreed += memList[i].size;
            memList[i].isFree = 1;
            printf("\n[INFO] Freed %zu bytes from '%s'\n", memList[i].size, name);
            return;
        }
    }
    printf("\n[WARNING] Variable '%s' not found or already freed.\n", name);
}
void showReport() {
    printf("\n========== MEMORY USAGE REPORT ==========\n");
    printf("Total Memory Allocated : %zu bytes\n", totalGiven);
    printf("Total Memory Freed     : %zu bytes\n", totalFreed);
    if (totalGiven > totalFreed) {
        printf("Memory Leak Detected   : %zu bytes not freed\n", totalGiven - totalFreed);
```

7. ***Code Implementation(continued)***

CPROJECT.c > main()

```c
void showReport() {
    if (totalGiven > totalFreed) {
        printf("Memory Leak Detected   : %zu bytes not freed\n", totalGiven - totalFreed);
        printf("\nLeaked Blocks:\n");
        for (int i = 0; i < memCount; i++) {
            if (!memList[i].isFree) {
                printf("  - '%s' at %p (%zu bytes)\n", memList[i].name, memList[i].ptr, memList[i].size);
            }
        }
    } else {
        printf("All memory successfully freed. No leaks detected.\n");
    }
    printf("=========================================\n");
}
void showMenu() {
    printf("\n==================================\n");
    printf("         MEMORY TRACKER MENU        \n");
    printf("==================================\n");
    printf("1. Allocate Memory\n");
    printf("2. Free Memory\n");
    printf("3. Show Memory Report\n");
    printf("0. Exit\n");
    printf("==================================\n");
    printf("Enter your choice: ");
}
int main() {
    int choice;
    printf("=========================================\n");
    printf("        Welcome to the Memory Tracker       \n");
    printf("  Monitor allocations and detect memory leaks\n");
    printf("=========================================\n");
    do {
        showMenu();
        scanf("%d", &choice);
        getchar();

        if (choice == 1) {
            char name[50];
            int size;
            printf("\nEnter variable name: ");
            fgets(name, sizeof(name), stdin);
            name[strcspn(name, "\n")] = 0;
            printf("Enter size in bytes: ");
            scanf("%d", &size);
            getchar();
            giveMem(size, name);
        } else if (choice == 2) {
            char name[50];
```



```
92          } else if (choice == 2) {
93              char name[50];
94              printf("\nEnter variable name to free: ");
95              fgets(name, sizeof(name), stdin);
96              name[strcspn(name, "\n")] = 0;
97              freeMem(name);
98          } else if (choice == 3) {
99              showReport();
100         } else if (choice != 0) {
101             printf("\n[ERROR] Invalid option. Please try again.\n");
102         }
103     } while (choice != 0);
104     printf("\nExiting program and displaying final report...\n");
105     showReport();
106     return 0;
107 }
```

8. ***Result Analysis***

The program has been tested and successfully handles the following:

- Accurate tracking of memory allocations and deallocations.
- Prevents memory from being freed twice.
- Reports all unfreed memory with exact sizes and pointers.
- Generates a useful summary at the end of each session.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
==================================
         MEMORY TRACKER MENU
==================================
1. Allocate Memory
2. Free Memory
3. Show Memory Report
0. Exit
==================================
Enter your choice: 3

========== MEMORY USAGE REPORT ==========
Total Memory Allocated : 13400001 bytes
Total Memory Freed     : 0 bytes
Memory Leak Detected   : 13400001 bytes not freed

Leaked Blocks:
  - 'targetvariable Y' at 00AB7020 (13400001 bytes)
==========================================

==================================
         MEMORY TRACKER MENU
==================================
1. Allocate Memory
2. Free Memory
3. Show Memory Report
0. Exit
==================================
Enter your choice: 1

Enter variable name: variable x
Enter size in bytes: 122

[INFO] Allocated 122 bytes for 'variable x' at address 008A1468

==================================
         MEMORY TRACKER MENU
==================================
1. Allocate Memory
2. Free Memory
3. Show Memory Report
0. Exit
==================================
Enter your choice: 3

========== MEMORY USAGE REPORT ==========
Total Memory Allocated : 13400123 bytes
Total Memory Freed     : 0 bytes
Memory Leak Detected   : 13400123 bytes not freed

Leaked Blocks:
  - 'targetvariable Y' at 00AB7020 (13400001 bytes)
  - 'variable x' at 008A1468 (122 bytes)
==========================================
```

```
========================================

==================================
         MEMORY TRACKER MENU
==================================
1. Allocate Memory
2. Free Memory
3. Show Memory Report
0. Exit
==================================
Enter your choice: 2

Enter variable name to free: variable x

[INFO] Freed 122 bytes from 'variable x'

==================================
         MEMORY TRACKER MENU
==================================
1. Allocate Memory
2. Free Memory
3. Show Memory Report
0. Exit
==================================
Enter your choice: 0

Exiting program and displaying final report...

========== MEMORY USAGE REPORT ==========
Total Memory Allocated : 13400123 bytes
Total Memory Freed     : 122 bytes
Memory Leak Detected   : 13400001 bytes not freed

Leaked Blocks:
  - 'targetvariable Y' at 00AB7020 (13400001 bytes)
========================================
PS C:\CODE\C,CPP>
```

9. Key Learnings

- Enhanced understanding of dynamic memory allocation and deallocation.
- Learned to manage and track memory in real time using structures and arrays.
- Gained proper understanding of structures and pointers separately as well as both combined together
- Making something meaningful not like any management systems etc
- Developed a deep understanding of low level memory allocation and memory management as well as its tracking .