Memory Leakage Detector for C Applications

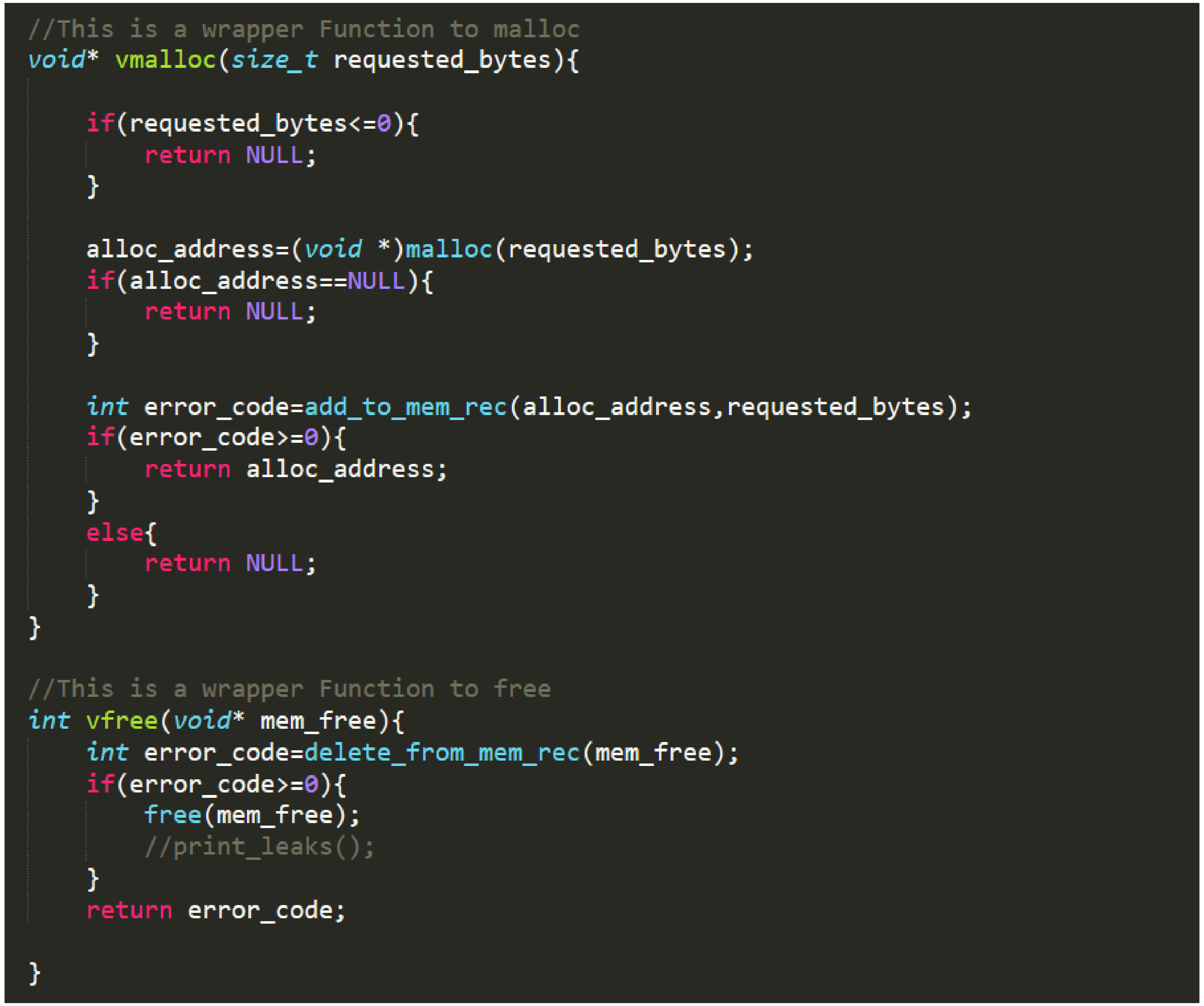
Git Hub Project Link:
https://github.com/lazyswan/C-Memory-Leakage-Detector

Swanand Sapre

FEATURES

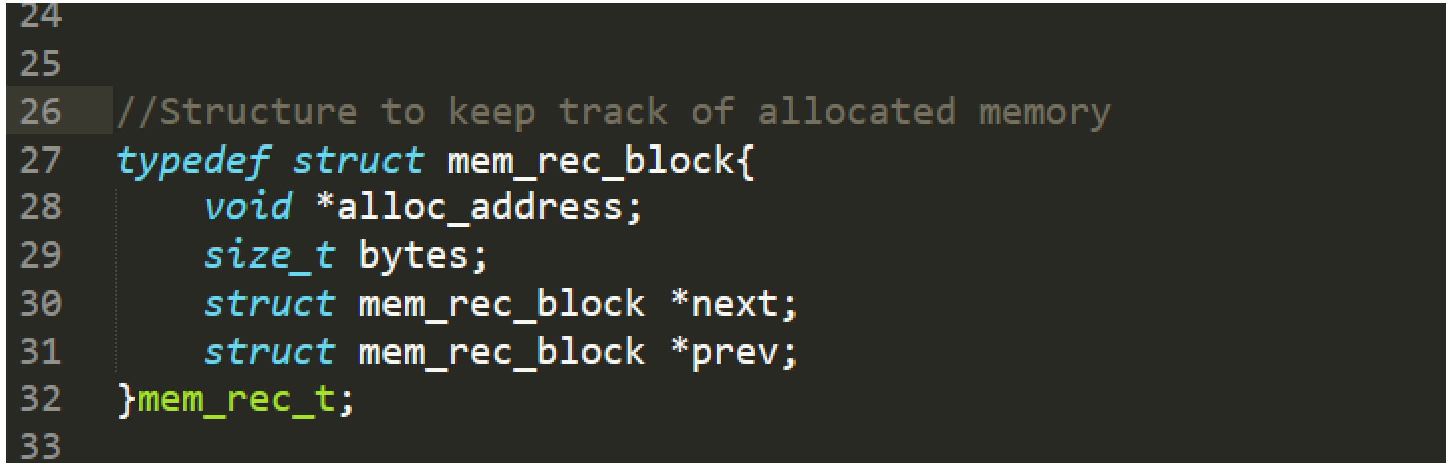
- Detects the Memory Leaks in any C-application.
- Time Complexity: O(n)
- Handles Abnormal Termination of Application as well as Normal Termination.

Wrapper Function to standard Malloc and Free Calls



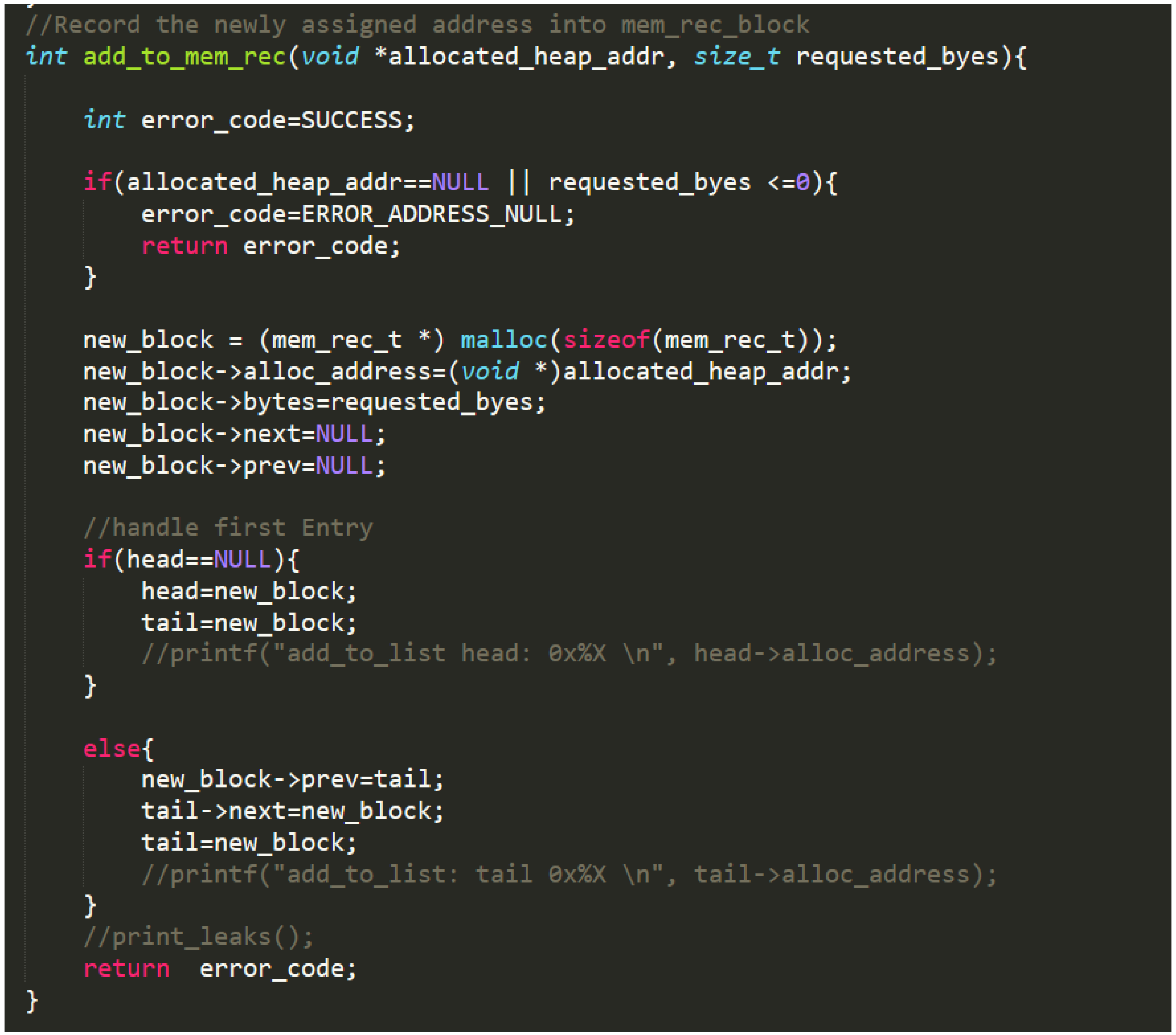
```c
//This is a wrapper Function to malloc
void* vmalloc(size_t requested_bytes){

    if(requested_bytes<=0){
        return NULL;
    }

    alloc_address=(void *)malloc(requested_bytes);
    if(alloc_address==NULL){
        return NULL;
    }

    int error_code=add_to_mem_rec(alloc_address,requested_bytes);
    if(error_code>=0){
        return alloc_address;
    }
    else{
        return NULL;
    }
}

//This is a wrapper Function to free
int vfree(void* mem_free){
    int error_code=delete_from_mem_rec(mem_free);
    if(error_code>=0){
        free(mem_free);
        //print_leaks();
    }
    return error_code;

}
```

New Doubly Linked List based data structure to store the records of dynamically allocated memory.



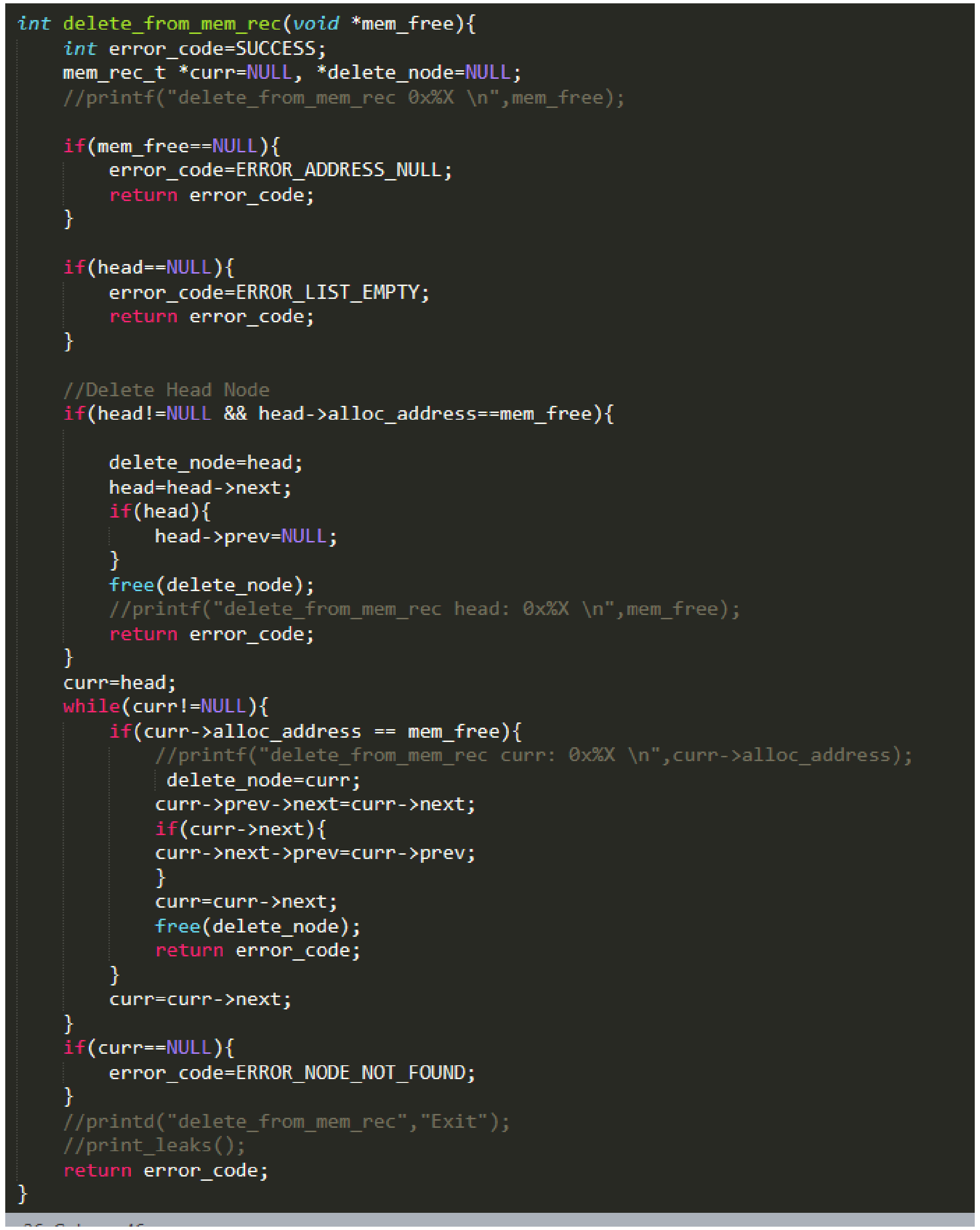
```


//Structure to keep track of allocated memory
typedef struct mem_rec_block{
    void *alloc_address;
    size_t bytes;
    struct mem_rec_block *next;
    struct mem_rec_block *prev;
}mem_rec_t;

```

The *vmalloc()* wrapper function maintains the record of allocated heap memory using *add_to_mem_rec()* function



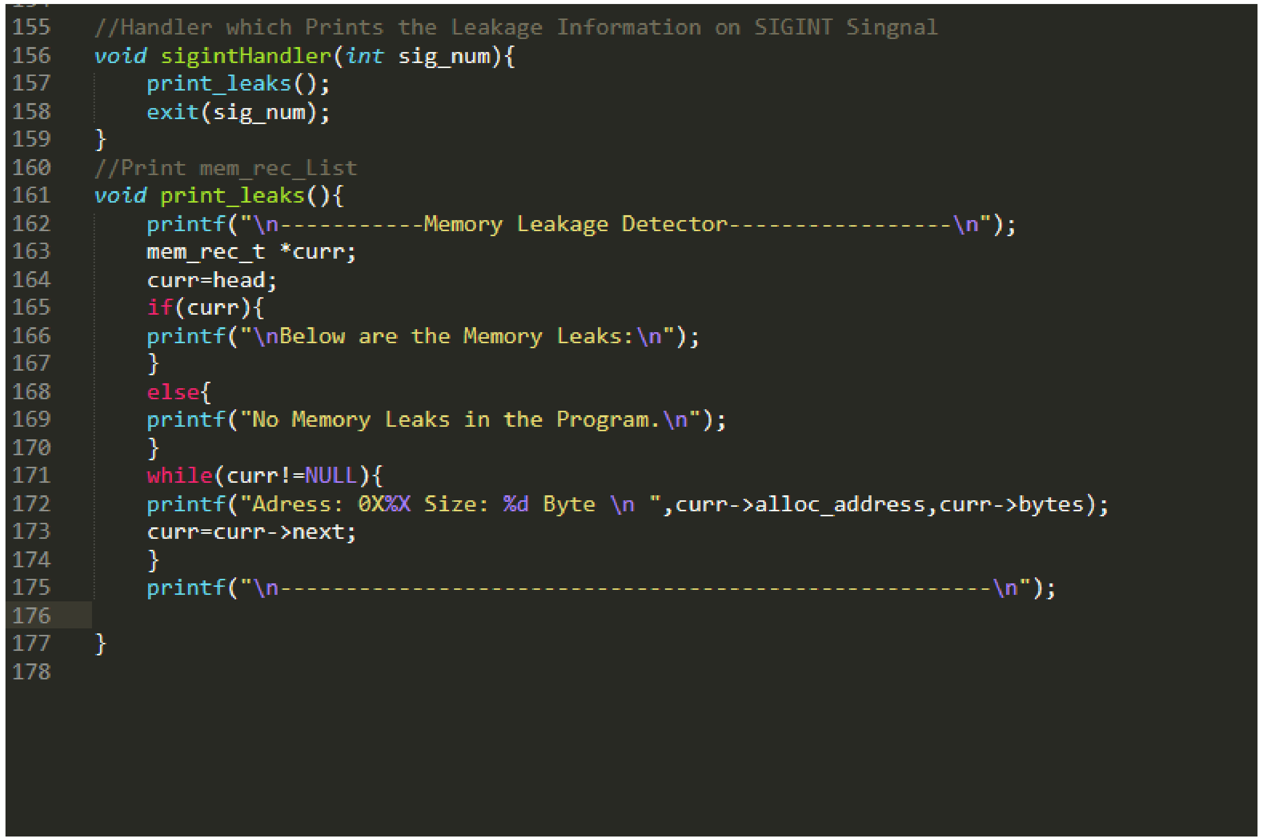
```
//Record the newly assigned address into mem_rec_block
int add_to_mem_rec(void *allocated_heap_addr, size_t requested_byes){

    int error_code=SUCCESS;

    if(allocated_heap_addr==NULL || requested_byes <=0){
        error_code=ERROR_ADDRESS_NULL;
        return error_code;
    }

    new_block = (mem_rec_t *) malloc(sizeof(mem_rec_t));
    new_block->alloc_address=(void *)allocated_heap_addr;
    new_block->bytes=requested_byes;
    new_block->next=NULL;
    new_block->prev=NULL;

    //handle first Entry
    if(head==NULL){
        head=new_block;
        tail=new_block;
        //printf("add_to_list head: 0x%X \n", head->alloc_address);
    }

    else{
        new_block->prev=tail;
        tail->next=new_block;
        tail=new_block;
        //printf("add_to_list: tail 0x%X \n", tail->alloc_address);
    }
    //print_leaks();
    return  error_code;
}
```

The *vfree()* wrapper function free the allocated heap memory as well as deletes the record from mem_rec list using *delete_from_mem_rec()* function



```c
int delete_from_mem_rec(void *mem_free){
    int error_code=SUCCESS;
    mem_rec_t *curr=NULL, *delete_node=NULL;
    //printf("delete_from_mem_rec 0x%X \n",mem_free);

    if(mem_free==NULL){
        error_code=ERROR_ADDRESS_NULL;
        return error_code;
    }

    if(head==NULL){
        error_code=ERROR_LIST_EMPTY;
        return error_code;
    }

    //Delete Head Node
    if(head!=NULL && head->alloc_address==mem_free){

        delete_node=head;
        head=head->next;
        if(head){
            head->prev=NULL;
        }
        free(delete_node);
        //printf("delete_from_mem_rec head: 0x%X \n",mem_free);
        return error_code;
    }
    curr=head;
    while(curr!=NULL){
        if(curr->alloc_address == mem_free){
            //printf("delete_from_mem_rec curr: 0x%X \n",curr->alloc_address);
             delete_node=curr;
            curr->prev->next=curr->next;
            if(curr->next){
            curr->next->prev=curr->prev;
            }
            curr=curr->next;
            free(delete_node);
            return error_code;
        }
        curr=curr->next;
    }
    if(curr==NULL){
        error_code=ERROR_NODE_NOT_FOUND;
    }
    //printd("delete_from_mem_rec","Exit");
    //print_leaks();
    return error_code;
}
```

siginitHandler() is signal handler which handles the abnormal termination.
It prints the details of leak memory using *print_leaks()* function

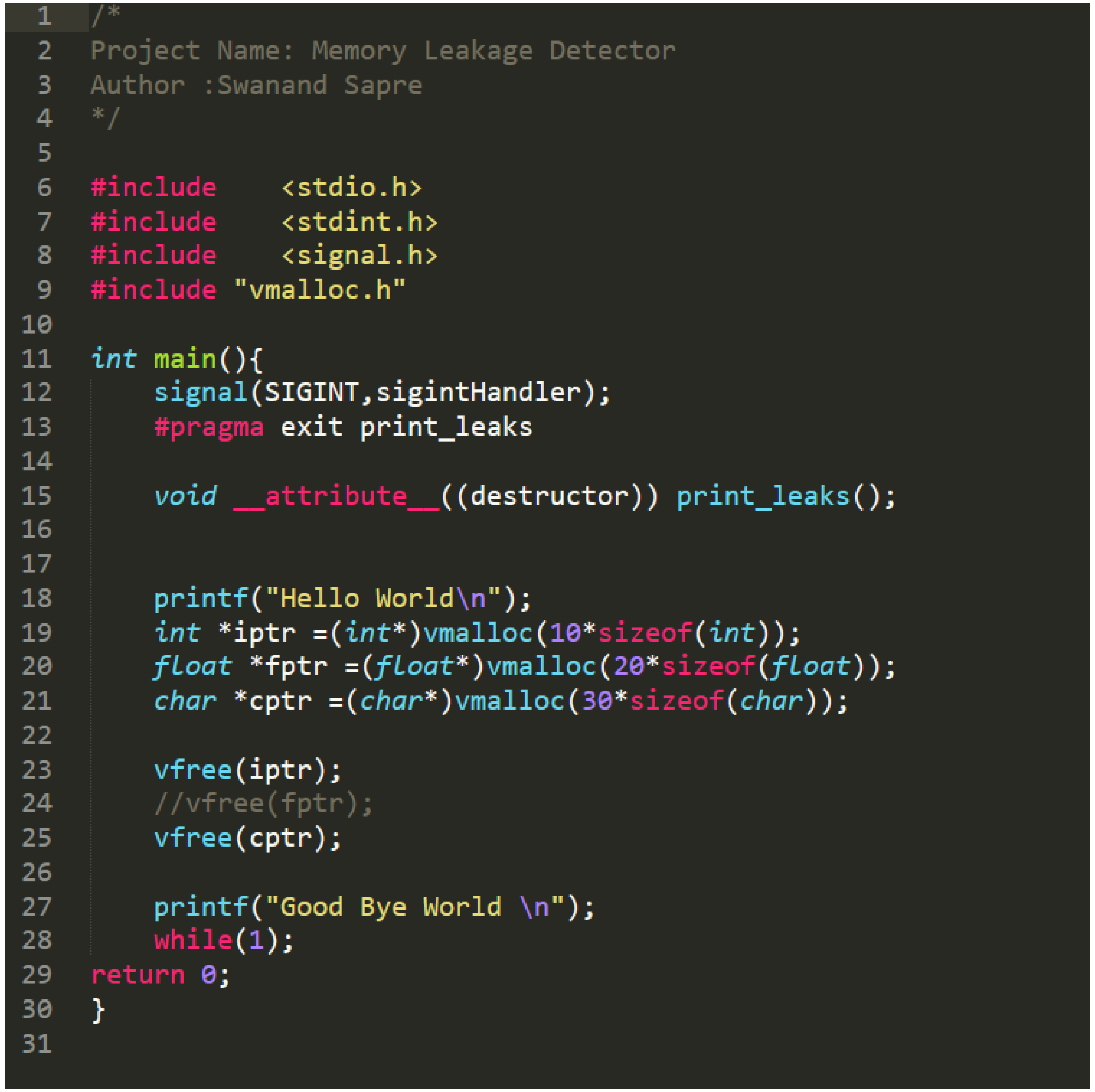


```
//Handler which Prints the Leakage Information on SIGINT Singnal
void sigintHandler(int sig_num){
    print_leaks();
    exit(sig_num);
}
//Print mem_rec_List
void print_leaks(){
    printf("\n-----------Memory Leakage Detector-----------------\n");
    mem_rec_t *curr;
    curr=head;
    if(curr){
    printf("\nBelow are the Memory Leaks:\n");
    }
    else{
    printf("No Memory Leaks in the Program.\n");
    }
    while(curr!=NULL){
    printf("Adress: 0X%X Size: %d Byte \n ",curr->alloc_address,curr->bytes);
    curr=curr->next;
    }
    printf("\n------------------------------------------------------\n");

}
```

Main Function

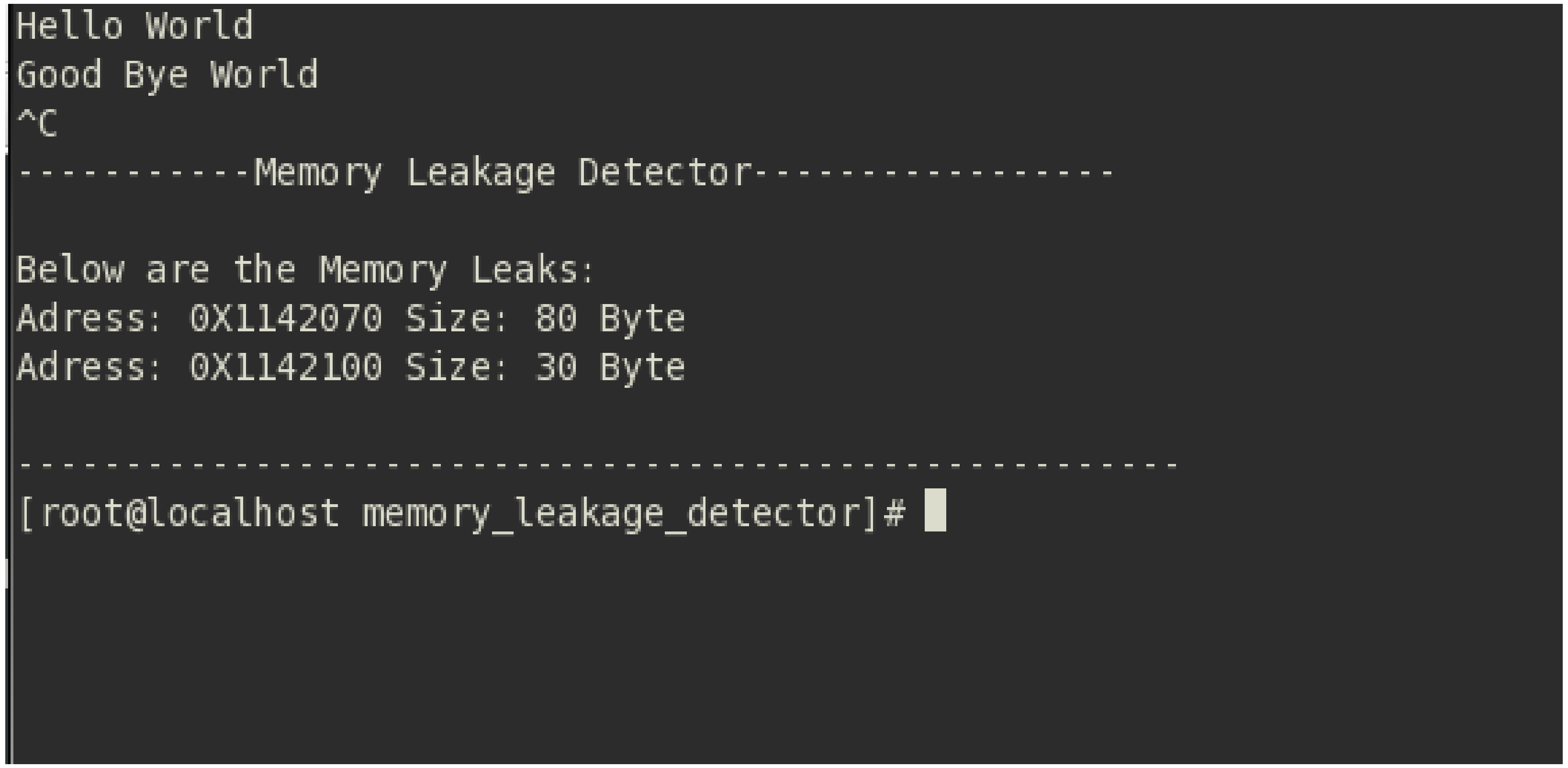
#pragma exit print_leaks

Executes the print_leaks()
And displays the memory leaks during normal termination of application.



```c
/*
Project Name: Memory Leakage Detector
Author :Swanand Sapre
*/

#include    <stdio.h>
#include    <stdint.h>
#include    <signal.h>
#include "vmalloc.h"

int main(){
    signal(SIGINT,sigintHandler);
    #pragma exit print_leaks

    void __attribute__((destructor)) print_leaks();


    printf("Hello World\n");
    int *iptr =(int*)vmalloc(10*sizeof(int));
    float *fptr =(float*)vmalloc(20*sizeof(float));
    char *cptr =(char*)vmalloc(30*sizeof(char));

    vfree(iptr);
    //vfree(fptr);
    vfree(cptr);

    printf("Good Bye World \n");
    while(1);
return 0;
}

```

OUTPUT



```
Hello World
Good Bye World
^C
-----------Memory Leakage Detector-----------------

Below are the Memory Leaks:
Adress: 0X1142070 Size: 80 Byte
Adress: 0X1142100 Size: 30 Byte

-----------------------------------------------------
[root@localhost memory_leakage_detector]#
```