VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum – 590 018

A project report on

"GARBAGE COLLECTOR IN C"

Submitted in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

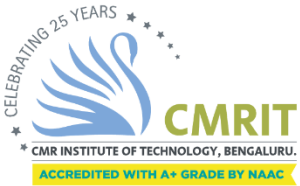
INFORMATION SCIENCE & ENGINEERING

by

SAHIL KUMAR RAO (1CR16IS088)
KEERTHANA C.V. (1CR16IS040)

Under the guidance of

Dr. M Farida Begam
Professor
Dept. of ISE, CMRIT, Bengaluru



CMR INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

#132, AECS Layout, IT Park Road, Bengaluru-560037

2019-20

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum – 590 018

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Certificate

This is to certify that the project entitled, **"Garbage Collector in C"**, is a bonafide work carried out by **Keerthana C.V. (1CR16IS040), Sahil Kumar Rao(1CR16IS088)** in partial fulfillment of the award of the degree of Bachelor of Engineering in Information Science & Engineering of Visvesvaraya Technological University, Belgaum, during the year 2019-20. It is certified that all corrections/suggestions indicated during reviews have been incorporated in the report. The project report satisfies the academic requirements in respect of the Phase II project work prescribed for the said Degree.

Name & Signature of Guide	**Name & Signature of HOD**
Dr. M Farida Begam	**Dr. M Farida Begam**
Professor	**HoD**
Dept. of ISE, CMRIT	**Dept. of ISE, CMRIT**

External Viva

Name of the examiners	Signature with date
1.	
2.	

CMR INSTITUTE OF TECHNOLOGY
BANGALORE-560037

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Declaration

We, **Keerthana C.V. (1CR16IS040), Sahil Kumar Rao(1CR16IS088)** bonafide students of CMR Institute of Technology, Bangalore, hereby declare that the dissertation entitled, **"Garbage Collector in C"** has been carried out by us under the guidance of Ms. Sheetal R, Associate Professor, CMRIT, Bangalore, in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Science Engineering, of the Visvesvaraya Technological University, Belgaum during the academic year 2019-2020. The work done in this dissertation report is original and it has not been submitted for any other degree in any university.

SAHIL KUMAR RAO (1CR16IS088)

KEERTHANA C. V. (1CR16IS040)

Acknowledgement

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of people who made it possible. Success is the epitome of hard work and perseverance, but steadfast of all is encouraging guidance.

So, it is with gratitude that we acknowledge all those whose guidance and encouragement served as beacon of light and crowned our effort with success.

We would like to thank Dr. Sanjay Jain, Principal, CMRIT, Bangalore, for providing an excellent academic environment in the college and his never-ending support for the B.E program.

We would like to express our gratitude towards Dr. Farida Begam, Professor and HOD, Department of Information Science & Engineering CMRIT, Bangalore, who provided guidance and gave valuable suggestions regarding the project.

We consider it a privilege and honor to express our sincere gratitude to our internal guide Dr. M Farida Begam, Professor, Department of Information Science & Engineering, CMRIT, Bangalore, for their valuable guidance throughout the tenure of this project work.

We would like to thank all the faculty members who have always been very cooperative and generous. Conclusively, we also thank all the non- teaching staff and all others who have done immense help directly or indirectly during our project.

SAHIL KUMAR RAO (1CR16IS088)

KEERTHANA C. V. (1CR16IS040)

TABLE OF CONTENT

Chapter 7

ABSTRACT

The aim of the project is to develop a garbage collector in 'C' language such that memory deallocation can be automated and problems like memory management can be handled from the compiler end instead of the user end. Continuous monitoring of the various data type pointers created, their registration and finally deallocation if required.

Although a developer can consider himself a responsible programmer who de-allocates the memory once allocated, yet there still remains a margin of error that can even compromise a whole system. This library-based application will be useful for many applications in the embedded world.

Chapter 1

PREAMBLE

1.1 Introduction

Since the advent of C/C++ Programming language, Memory management is one of the responsibilities which the developer has to deal with

C/C++ Software often suffers from Two Memory related Problems:

Memory corruption

Memory leak

Unlike Java, C/C++ does not have the luxury for automatic garbage collection.

Java does not allow programmer to access the physical memory directly, but C/C++ does. Therefore, Java applications do not suffer from Memory corruption either, but C/C++ does

Here, we will design and implement **Garbage Collection tool** for C programs which can be extended to use in C++

1.2 Problem Statement

The aim of the project is to develop a garbage collector in 'C' language such that memory deallocation can be automated and problems like memory management can be handled from the compiler end instead of the user end. Continuous monitoring of the various data type pointers created, their registration and finally deallocation if required.

Chapter 2

SYSTEM REQUIREMENTS SPECIFICATION

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

In order to fully understand one's project, it is very important that they come up with an SRS listing out their requirements, how are they going to meet it and how will they complete the project. SRS also functions as a blueprint for completing a project with as little cost growth as possible. SRS is often referred to as the parent document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

Requirement is a condition or capability to which the system must conform. Requirement Management is a systematic approach towards eliciting, organizing and documenting the requirements of the system clearly along with the applicable attributes. The elusive difficulties of requirements are not always obvious and can come from any number of sources.

2.1 System Requirement

Hardware System Configuration

- Processor - Any
- Memory – Minimum 256 MB
- Disk – Minimum 32 MB

Software System Configuration

- Operating System – Windows/Linux/MAC
- Compiler - GCC

Chapter 3

Literature Survey

3.1 ***"Research and analysis of garbage collection mechanism for embedded systems":*** IEEE Publications (Liu Wei; Chen Zhang-long; Tu Shi-hang):

This was the first paper that presented the idea behind use of garbage collection in embedded systems, hence C and C++ based systems. We got to know about the various places we need to implement garbage collection.

3.2 ***"Comparisons of garbage collector Prototypes for C++ applications":*** The 7th IEEE/ACS International Conference on Computer Systems and Applications***:***

This research is oriented toward the various possible ways of creating a garbage collector in C++/C and their Comparisons. Every method has its own drawbacks and criticalities. From here we got idea on our algorithm for the project.

3.3 ***"Comparisons of garbage collectors in java programming language":*** 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO):

In order to understand the working of most prospering garbage collector, we looked into this publication. We understood the reasons behind the working of java garbage collector and cloned the same behavior in our project.

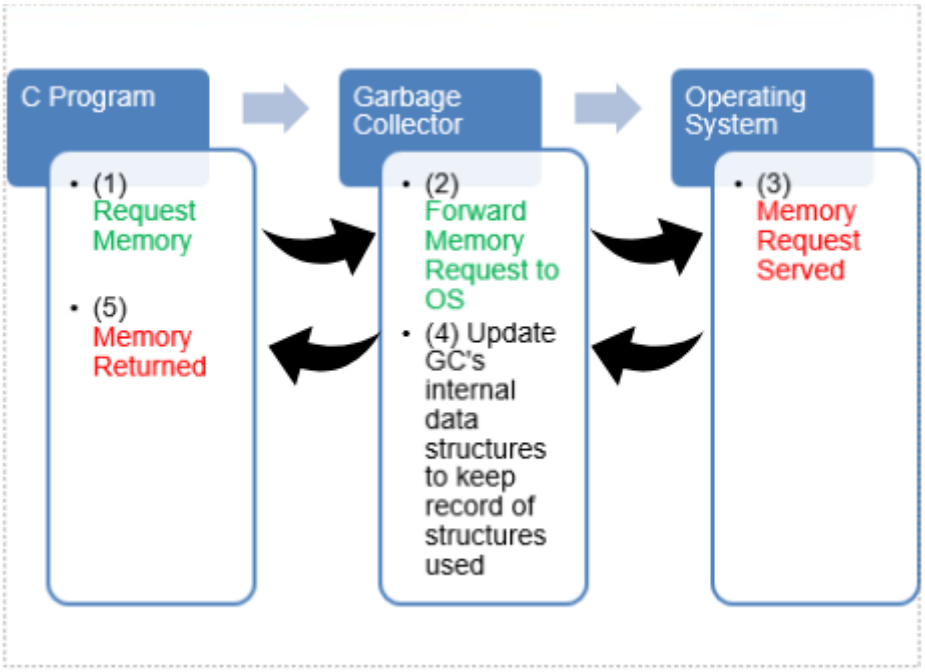
Chapter 4

SYSTEM APPROACH AND DESIGN

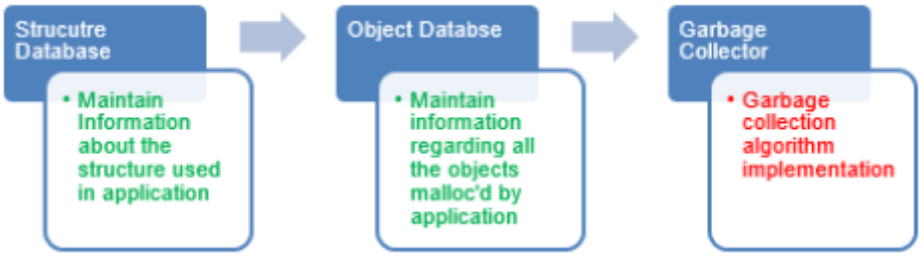
We have our project in three different modules. Each module has its implementation boundaries and specific goals assigned to it. These modules contribute to the overall approach and design of the project.

The 3 modules are as follows:

- Development of basic header and its relevant C file
- Developing the Structure database and Object Database
- Developing the Collection algorithm



Block Diagram



Module 1:

Our goals in this module are to make sure that every data type used in the application should be registered with its type and the amount of space an object of that type requires.
It's the responsibility of this module to:

- Register all the data types used in the application
- Application should give information related to the various structures that are created
- A linked-list to be maintained to save these structures related information
- We have used name of the structure as key for searching

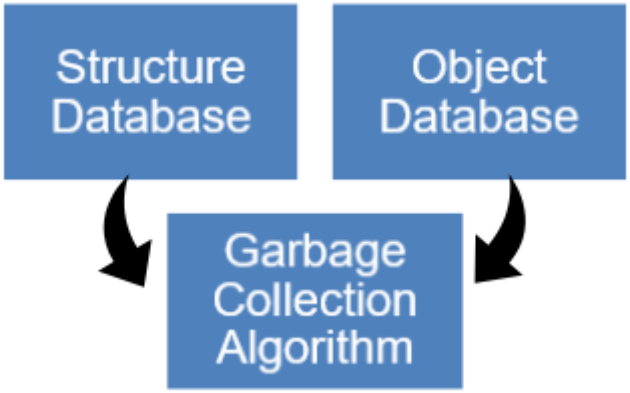
Module 2:

Our goals in this module are to make sure that every object that is called by the application for memory should be registered with its type and the address of allocation.
It's the responsibility of this module to:

- Register all the pointers/objects used in the application
- A linked-list to be maintained to save these structures related information
- Every node has address and structure information related to the object created

Module 3:

Our goal in this module is to process the object database, with the help of structure database and find the objects that are de-allocated/leaked. Our Garbage collection algorithm works on the Disjoint set arrangement of the data types used in any application.
Whenever we write an application, all the data types present in it are automatically arranged in a disjoint/Directed cyclic set formation.
If any object is not reachable by the root of this graph, it will be eligible for garbage collection. Hence, our garbage collector problem is basically a graph problem to find out non-reachable nodes and using the object database, we can free those pointers which are no longer reachable by the root.


Structure Database
Object Database
Garbage Collection Algorithm

Chapter 5

Implementation

Header File for Garbage Collector: (gc.h)

```
#ifndef __MLD__
#include <assert.h>
#include <string.h>

/*Structure Data base Definition Begin*/

#define MAX_STRUCTURE_NAME_SIZE 128
#define MAX_FIELD_NAME_SIZE 128

/*Enumeration for data types*/
typedef enum {
   UINT8,
   UINT32,
   INT32,
   CHAR,
   OBJ_PTR,
   VOID_PTR, /*New Data type added to identify void * pointers*/
   FLOAT,
   DOUBLE,
   OBJ_STRUCT
} data_type_t;

typedef enum{

   MLD_FALSE,
   MLD_TRUE
} mld_boolean_t;



#define OFFSETOF(struct_name, fld_name)     \
   (unsigned int)&(((struct_name *)0)->fld_name)

#define FIELD_SIZE(struct_name, fld_name)   \
   sizeof(((struct_name *)0)->fld_name)


typedef struct _struct_db_rec_ struct_db_rec_t;

/*Structure to store the information of one field of a
* C structure*/
```

```
typedef struct _field_info_{
   char fname [MAX_FIELD_NAME_SIZE];   /*Name of the field*/
   data_type_t dtype;                  /*Data type of the field*/
   unsigned int size;                 /*Size of the field*/
   unsigned int offset;                /*Offset of the field*/
   // Below field is meaningful only if dtype = OBJ_PTR, Or OBJ_STRUCT
   char nested_str_name[MAX_STRUCTURE_NAME_SIZE];
} field_info_t;

/*Structure to store the information of one C structure
* which could have 'n_fields' fields*/
struct _struct_db_rec_{
   struct_db_rec_t *next;  /*Pointer to the next structure in the linked list*/
   char struct_name [MAX_STRUCTURE_NAME_SIZE];  // key
   unsigned int ds_size;   /*Size of the structure*/
   unsigned int n_fields;  /*No of fields in the structure*/
   field_info_t *fields;   /*pointer to the array of fields*/
};

/*Finally the head of the linked list representing the structure
* database*/
typedef struct _struct_db_{
   struct_db_rec_t *head;
   unsigned int count;
} struct_db_t;


/*Structure Data base Definition Ends*/

/* Printing functions*/
void
print_structure_rec (struct_db_rec_t *struct_rec);

void
print_structure_db(struct_db_t *struct_db);

/* Fn to add the structure record in a structure database */

int /*return 0 on success, -1 on failure for some reason*/
add_structure_to_struct_db(struct_db_t *struct_db, struct_db_rec_t *struct_rec);

/*Structure Registration helping APIs*/

#define FIELD_INFO(struct_name, fld_name, dtype, nested_struct_name)    \
  {#fld_name, dtype, FIELD_SIZE(struct_name, fld_name),              \
      OFFSETOF(struct_name, fld_name), #nested_struct_name}

#define REG_STRUCT(struct_db, st_name, fields_arr)                  \
   do{                                                \
      struct_db_rec_t *rec = calloc(1, sizeof(struct_db_rec_t));    \
```

```
        strncpy(rec->struct_name, #st_name, MAX_STRUCTURE_NAME_SIZE); \
        rec->ds_size = sizeof(st_name);                               \
        rec->n_fields = sizeof(fields_arr)/sizeof(field_info_t);      \
        rec->fields = fields_arr;                                     \
        if(add_structure_to_struct_db(struct_db, rec)){               \
           assert(0);                                                 \
        }                                                             \
    }while(0);

/*Structure Data base Definition Ends*/


/*Object Database structure definitions Starts here*/

typedef struct _object_db_rec_ object_db_rec_t;

struct _object_db_rec_{
    object_db_rec_t *next;
    void *ptr;
    unsigned int units;
    struct_db_rec_t *struct_rec;
    mld_boolean_t is_visited; /*Used for Graph traversal*/
    mld_boolean_t is_root;    /*Is this object is Root object*/
};

typedef struct _object_db_{
    struct_db_t *struct_db;
    object_db_rec_t *head;
    unsigned int count;
} object_db_t;


/*Dumping functions*/
void
print_object_rec(object_db_rec_t *obj_rec, int i);

void
print_object_db(object_db_t *object_db);

/*API to malloc the object*/
void*
xcalloc(object_db_t *object_db, char *struct_name, int units);

/*APIs to register root objects*/
void mld_register_root_object (object_db_t *object_db,
                               void *objptr,
                               char *struct_name,
                               unsigned int units);

void
```

```
set_mld_object_as_global_root(object_db_t *object_db, void *obj_ptr);


/*APIs for MLD Algorithm*/
void
run_mld_algorithm(object_db_t *object_db);

void
report_leaked_objects(object_db_t *object_db);

#endif /* __MLD__ */
```

C file for Garbage Collector: (gc.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "mld.h"
#include "css.h"
#include <assert.h>
#include <memory.h>

char *DATA_TYPE[] = {"UINT8", "UINT32", "INT32",
                     "CHAR", "OBJ_PTR", "VOID_PTR", "FLOAT",
                     "DOUBLE", "OBJ_STRUCT"};

/* Dumping Function */

void
print_structure_rec(struct_db_rec_t *struct_rec){
   if(!struct_rec) return;
   int j = 0;
   field_info_t *field = NULL;
   printf(ANSI_COLOR_CYAN                "|-----------------------------------------------------|\n"
ANSI_COLOR_RESET);
   printf(ANSI_COLOR_YELLOW  "| %-20s | size = %-8d | #flds = %-3d |\n"
ANSI_COLOR_RESET, struct_rec->struct_name, struct_rec->ds_size, struct_rec->n_fields);
   printf(ANSI_COLOR_CYAN  "|-----------------------------------------------------|-----------------
-----------------------------------------------------------------------|\n" ANSI_COLOR_RESET);
   for(j = 0; j < struct_rec->n_fields; j++){
      field = &struct_rec->fields[j];
      printf("  %-20s |", "");
      printf("%-3d %-20s | dtype = %-15s | size = %-5d | offset = %-6d|  nstructname = %-20s
|\n",
            j, field->fname, DATA_TYPE[field->dtype], field->size, field->offset, field-
>nested_str_name);
      printf("  %-20s |", "");
      printf(ANSI_COLOR_CYAN  "-----------------------------------------------------------------
-----------------------------------------------------|\n" ANSI_COLOR_RESET);
   }
```

```
}

void
print_structure_db(struct_db_t *struct_db){

    if(!struct_db) return;
    printf("printing STRUCURE DATABASE\n");
    int i = 0;
    struct_db_rec_t *struct_rec = NULL;
    struct_rec = struct_db->head;
    printf("No of Structures Registered = %d\n", struct_db->count);
    while(struct_rec){
        printf("structure No : %d (%p)\n", i++, struct_rec);
        print_structure_rec(struct_rec);
        struct_rec = struct_rec->next;
    }
}

int
add_structure_to_struct_db(struct_db_t *struct_db,
                           struct_db_rec_t *struct_rec){

    struct_db_rec_t *head = struct_db->head;

    if(!head){
        struct_db->head = struct_rec;
        struct_rec->next = NULL;
        struct_db->count++;
        return 0;
    }

    struct_rec->next = head;
    struct_db->head = struct_rec;
    struct_db->count++;
    return 0;
}

static struct_db_rec_t *
struct_db_look_up(struct_db_t *struct_db,
                  char *struct_name){

    struct_db_rec_t *head = struct_db->head;
    if(!head) return NULL;

    for(; head; head = head->next){
        if(strncmp(head->struct_name, struct_name, MAX_STRUCTURE_NAME_SIZE) ==0)
            return head;
    }
    return NULL;
}
```

```
static object_db_rec_t *
object_db_look_up(object_db_t *object_db, void *ptr){

    object_db_rec_t *head = object_db->head;
    if(!head) return NULL;

    for(; head; head = head->next){
        if(head->ptr == ptr)
            return head;
    }
    return NULL;
}

/*Working with objects*/
static void
add_object_to_object_db(object_db_t *object_db,
                        void *ptr,
                        int units,
                        struct_db_rec_t *struct_rec,
                        mld_boolean_t is_root){

    object_db_rec_t *obj_rec = object_db_look_up(object_db, ptr);
    /*Dont add same object twice*/
    assert(!obj_rec);

    obj_rec = calloc(1, sizeof(object_db_rec_t));

    obj_rec->next = NULL;
    obj_rec->ptr = ptr;
    obj_rec->units = units;
    obj_rec->struct_rec = struct_rec;
    obj_rec->is_visited = MLD_FALSE;
    obj_rec->is_root = is_root;

    object_db_rec_t *head = object_db->head;

    if(!head){
        object_db->head = obj_rec;
        obj_rec->next = NULL;
        object_db->count++;
        return;
    }

    obj_rec->next = head;
    object_db->head = obj_rec;
    object_db->count++;
}
```

```
void *
xcalloc(object_db_t *object_db,
        char *struct_name,
        int units){

    struct_db_rec_t *struct_rec = struct_db_look_up(object_db->struct_db, struct_name);
    assert(struct_rec);
    void *ptr = calloc(units, struct_rec->ds_size);
    add_object_to_object_db(object_db, ptr, units, struct_rec, MLD_FALSE);   /*xcalloc by
default set the object as non-root*/
    return ptr;
}

static void
delete_object_record_from_object_db(object_db_t *object_db,
                                     object_db_rec_t *object_rec){

    assert(object_rec);

    object_db_rec_t *head = object_db->head;
    if(head == object_rec){
        object_db->head = object_rec->next;
        free(object_rec);
        return;
    }

    object_db_rec_t *prev = head;
    head = head->next;

    while(head){
        if(head != object_rec){
            prev = head;
            head = head->next;
            continue;
        }

        prev->next = head->next;
        head->next = NULL;
        free(head);
        return;
    }
}


void
xfree(object_db_t *object_db, void *ptr){

    if(!ptr) return;
    object_db_rec_t *object_rec =
```

```
        object_db_look_up(object_db, ptr);

    assert(object_rec);
    assert(object_rec->ptr);
    free(object_rec->ptr);
    object_rec->ptr = NULL;
    /*Delete object record from object db*/
    delete_object_record_from_object_db(object_db, object_rec);
}

/*Dumping Functions for Object database*/
void
print_object_rec(object_db_rec_t *obj_rec, int i){

    if(!obj_rec) return;
    printf(ANSI_COLOR_MAGENTA "------------------------------------------------------------------
-----------------------------------|\n"ANSI_COLOR_RESET);
    printf(ANSI_COLOR_YELLOW "%-3d ptr = %-10p | next = %-10p | units = %-4d |
struct_name = %-10s | is_root = %s |\n"ANSI_COLOR_RESET,
        i, obj_rec->ptr, obj_rec->next, obj_rec->units, obj_rec->struct_rec->struct_name,
obj_rec->is_root ? "TRUE " : "FALSE");
    printf(ANSI_COLOR_MAGENTA "------------------------------------------------------------------
-----------------------------------|\n"ANSI_COLOR_RESET);
}

void
print_object_db(object_db_t *object_db){

    object_db_rec_t *head = object_db->head;
    unsigned int i = 0;
    printf(ANSI_COLOR_CYAN "Printing OBJECT DATABASE\n");
    for(; head; head = head->next){
        print_object_rec(head, i++);
    }
}

/*The global object of the application which is not created by xcalloc
* should be registered with MLD using below API*/
void
mld_register_global_object_as_root (object_db_t *object_db,
                    void *objptr,
                    char *struct_name,
                    unsigned int units){

    struct_db_rec_t *struct_rec = struct_db_look_up(object_db->struct_db, struct_name);
    assert(struct_rec);

   /*Create a new object record and add to object database*/
   add_object_to_object_db(object_db, objptr, units, struct_rec, MLD_TRUE);
}
```

```
/* Application might create an object using xcalloc , but at the same time the object
* can be root object. Use this API to override the object flags for the object already
* preent in object db*/
void
mld_set_dynamic_object_as_root(object_db_t *object_db, void *obj_ptr){

    object_db_rec_t *obj_rec = object_db_look_up(object_db, obj_ptr);
    assert(obj_rec);

    obj_rec->is_root = MLD_TRUE;
}


static object_db_rec_t *
get_next_root_object(object_db_t *object_db,
                     object_db_rec_t *starting_from_here){

    object_db_rec_t *first = starting_from_here ? starting_from_here->next : object_db->head;
    while(first){
        if(first->is_root)
            return first;
        first = first->next;
    }
    return NULL;
}

static void
init_mld_algorithm(object_db_t *object_db){

     object_db_rec_t *obj_rec = object_db->head;
     while(obj_rec){
         obj_rec->is_visited = MLD_FALSE;
         obj_rec = obj_rec->next;
     }
}

/* Level 2 Pseudocode : This function explore the direct childs of obj_rec and mark
* them visited. Note that obj_rec must have already visted.*/
static void
mld_explore_objects_recursively(object_db_t *object_db,
                                object_db_rec_t *parent_obj_rec){

    unsigned int i , n_fields;
    char *parent_obj_ptr = NULL,
         *child_obj_offset = NULL;
    void *child_object_address = NULL;
    field_info_t *field_info = NULL;

    object_db_rec_t *child_object_rec = NULL;
```

```
struct_db_rec_t *parent_struct_rec = parent_obj_rec->struct_rec;

/*Parent object must have already visited*/
assert(parent_obj_rec->is_visited);

if(parent_struct_rec->n_fields == 0){
   return;
}

for( i = 0; i < parent_obj_rec->units; i++){

   parent_obj_ptr = (char *)(parent_obj_rec->ptr) + (i * parent_struct_rec->ds_size);

   for(n_fields = 0; n_fields < parent_struct_rec->n_fields; n_fields++){

      field_info = &parent_struct_rec->fields[n_fields];

      /*We are only concerned with fields which are pointer to
       * other objects*/
      switch(field_info->dtype){
         case UINT8:
         case UINT32:
         case INT32:
         case CHAR:
         case FLOAT:
         case DOUBLE:
         case OBJ_STRUCT:
            break;
         case VOID_PTR:
         case OBJ_PTR:
         default:
            ;

         /*child_obj_offset is the memory location inside parent object
          * where address of next level object is stored*/
         child_obj_offset = parent_obj_ptr + field_info->offset;
         memcpy(&child_object_address, child_obj_offset, sizeof(void *));

         /*child_object_address now stores the address of the next object in the
          * graph. It could be NULL, Handle that as well*/
         if(!child_object_address) continue;

         child_object_rec = object_db_look_up(object_db, child_object_address);

         assert(child_object_rec);
         /* Since we are able to reach this child object "child_object_rec"
          * from parent object "parent_obj_ptr", mark this
          * child object as visited and explore its children recirsively.
          * If this child object is already visited, then do nothing - avoid infinite loops*/
         if(!child_object_rec->is_visited){
```

```
                child_object_rec->is_visited = MLD_TRUE;
                if(field_info->dtype != VOID_PTR) /*Explore next object only when it is not a
VOID_PTR*/
                    mld_explore_objects_recursively(object_db, child_object_rec);
            }
            else{
                continue; /*Do nothing, explore next child object*/
            }
        }
    }
  }
}

/* Level 1 Pseudocode : We will traverse the graph starting from root objects
* and mark all reachable nodes as visited*/
void
run_mld_algorithm(object_db_t *object_db){

   /*Step 1 : Mark all objects in object databse as unvisited*/
   init_mld_algorithm(object_db);

   /* Step 2 : Get the first root object from the object db, it could be
    * present anywhere in object db. If there are multiple roots in object db
    * return the first one, we can start mld algorithm from any root object*/

   object_db_rec_t *root_obj = get_next_root_object(object_db, NULL);

   while(root_obj){
      if(root_obj->is_visited){
         /* It means, all objects reachable from this root_obj has already been
          * explored, no need to do it again, else you will end up in infinite loop.
          * Remember, Application Data structures are cyclic graphs*/
         root_obj = get_next_root_object(object_db, root_obj);
         continue;
      }

      /*root objects are always reachable since application holds the global
       * variable to it*/
      root_obj->is_visited = MLD_TRUE;

      /*Explore all reachable objects from this root_obj recursively*/
      mld_explore_objects_recursively(object_db, root_obj);

      root_obj = get_next_root_object(object_db, root_obj);
   }
}


static void
mld_dump_object_rec_detail(object_db_rec_t *obj_rec){
```

```
    int n_fields = obj_rec->struct_rec->n_fields;
    field_info_t *field = NULL;

    int units = obj_rec->units, obj_index = 0,
        field_index = 0;

    for(; obj_index < units; obj_index++){
        char *current_object_ptr = (char *)(obj_rec->ptr) + \
                    (obj_index * obj_rec->struct_rec->ds_size);

        for(field_index = 0; field_index < n_fields; field_index++){

            field = &obj_rec->struct_rec->fields[field_index];

            switch(field->dtype){
                case UINT8:
                case INT32:
                case UINT32:
                    printf("%s[%d]->%s = %d\n", obj_rec->struct_rec->struct_name, obj_index,
field->fname, *(int *)(current_object_ptr + field->offset));
                    break;
                case CHAR:
                    printf("%s[%d]->%s = %s\n", obj_rec->struct_rec->struct_name, obj_index,
field->fname, (char *)(current_object_ptr + field->offset));
                    break;
                case FLOAT:
                    printf("%s[%d]->%s = %f\n", obj_rec->struct_rec->struct_name, obj_index,
field->fname, *(float *)(current_object_ptr + field->offset));
                    break;
                case DOUBLE:
                    printf("%s[%d]->%s = %f\n", obj_rec->struct_rec->struct_name, obj_index,
field->fname, *(double *)(current_object_ptr + field->offset));
                    break;
                case OBJ_PTR:
                    printf("%s[%d]->%s = %p\n", obj_rec->struct_rec->struct_name, obj_index,
field->fname,  (void *)*(int *)(current_object_ptr + field->offset));
                    break;
                case OBJ_STRUCT:
                    /*Later*/
                    break;
                default:
                    break;
            }
        }
    }
}


void
```

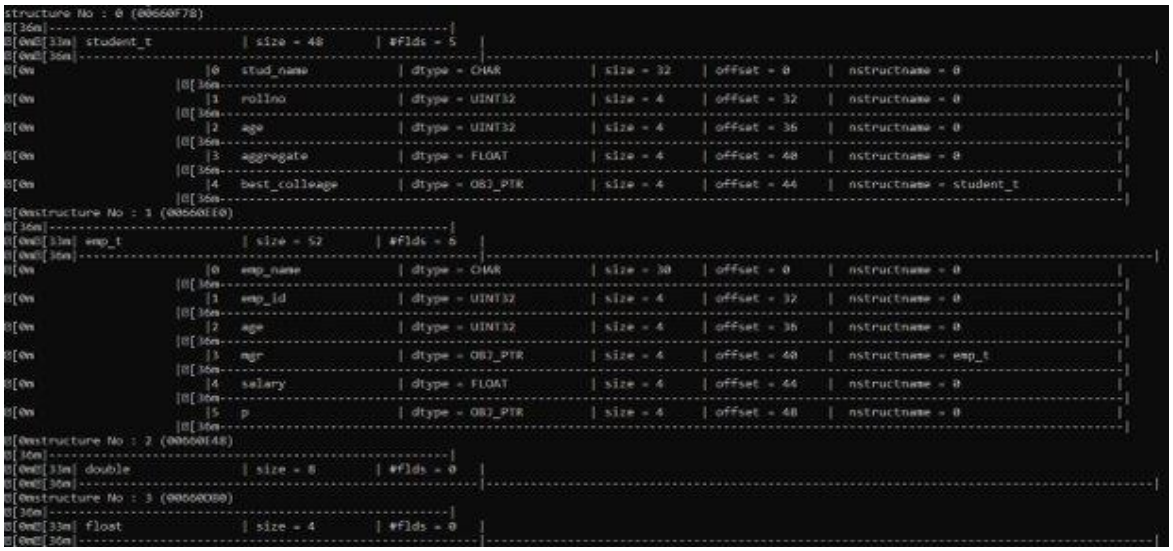
```
report_leaked_objects(object_db_t *object_db){

    int i = 0;
    object_db_rec_t *head;

    printf("Dumping Leaked Objects\n");

    for(head = object_db->head; head; head = head->next){
        if(!head->is_visited){
            print_object_rec(head, i++);
            mld_dump_object_rec_detail(head);
            printf("\n\n");
        }
    }
}


/*Support for primitive data types*/
void
mld_init_primitive_data_types_support(struct_db_t *struct_db){

    REG_STRUCT(struct_db, int , 0);
    REG_STRUCT(struct_db, float , 0);
    REG_STRUCT(struct_db, double , 0);
}
```

Appointments Pending

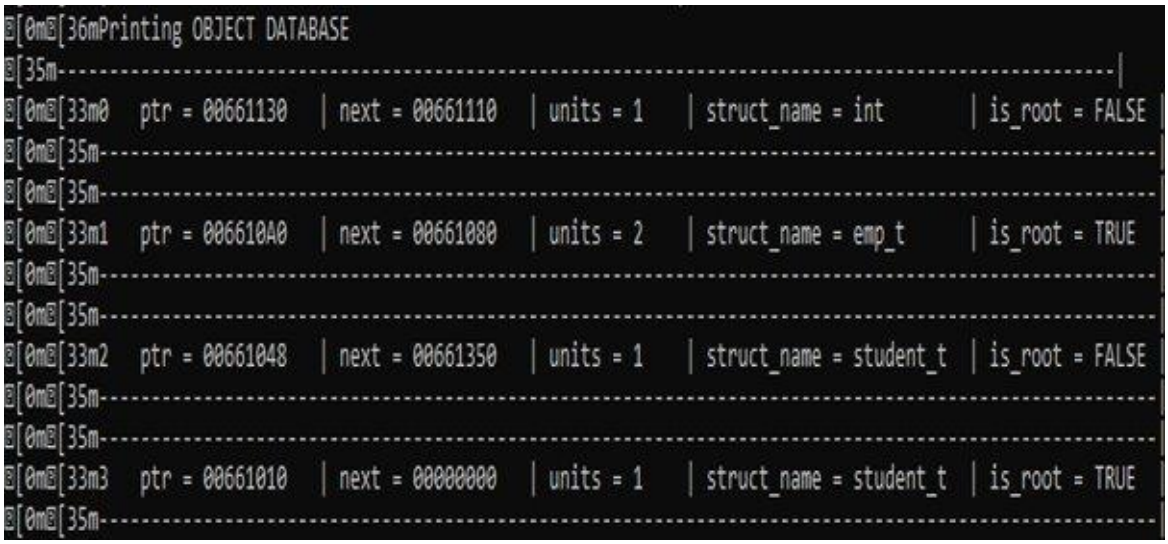
Chapter 6

Screenshots

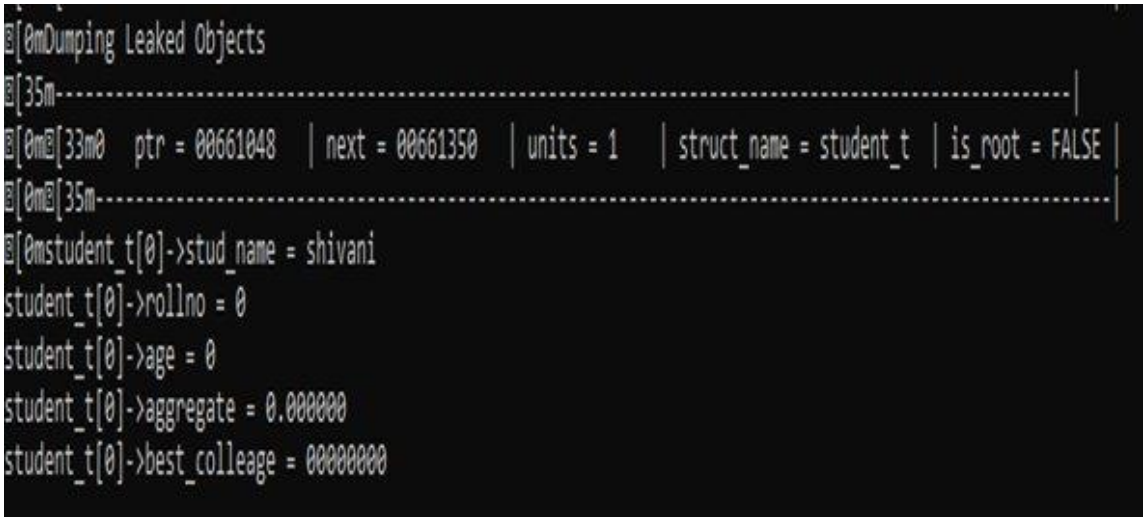
Module 1: Strucuture Database



Module 2: Object Database



Module 3: Collected Objects



Chapter 7

Conclusion

- Offers a garbage collection based on the reachability of the pointers that are created
- Close to the Java garbage collection in functionality
- Can be extended to use in C++ as well

References

- ***"Research and analysis of garbage collection mechanism for embedded systems"***
- ***"Comparisons of garbage collector Prototypes for C++ applications"***
- ***"Comparisons of garbage collectors in java programming language"***
- ***Reddit solutions***