EXPERIMENT NO :

December 3, 2023

1 NAME -; Tejal Gaikwad B35

2 NAME -: Asmita Bagal B37

Title:programs on garbage collection, packaging, access modifier , as well as static and abstract modifiers

Aim:

programs on garbage collection ,packaging , access modifiers , as well as static modifiers and abstract modifiers

Learning objectives:

1 Students should able to design and analyze simple linear and non linear data structures.

2It strengthen the ability to the students to identify and apply the suitable data structure for the given real world problem. **3** It enables them to gain knowledge in practical applications of data structures .

Theory:

Garbage Collection:

It automates memory management by reclaiming memory occupied by objects that are no longer in use. This prevents memory leaks and makes the development process more efficient.

Packaging:

Packaging, or organizing code into packages or namespaces, is crucial for code maintainability and organization. It allows developers to group related classes and functions, making it easier to manage large codebases. Good packaging practices help in code reuse and collaboration.

Access Modifiers:

Access modifiers (public, private, protected, default) control the visibility and accessibility of class members in object-oriented programming. They determine who can access or modify class fields and methods. Access modifiers play a significant role in encapsulation and security.

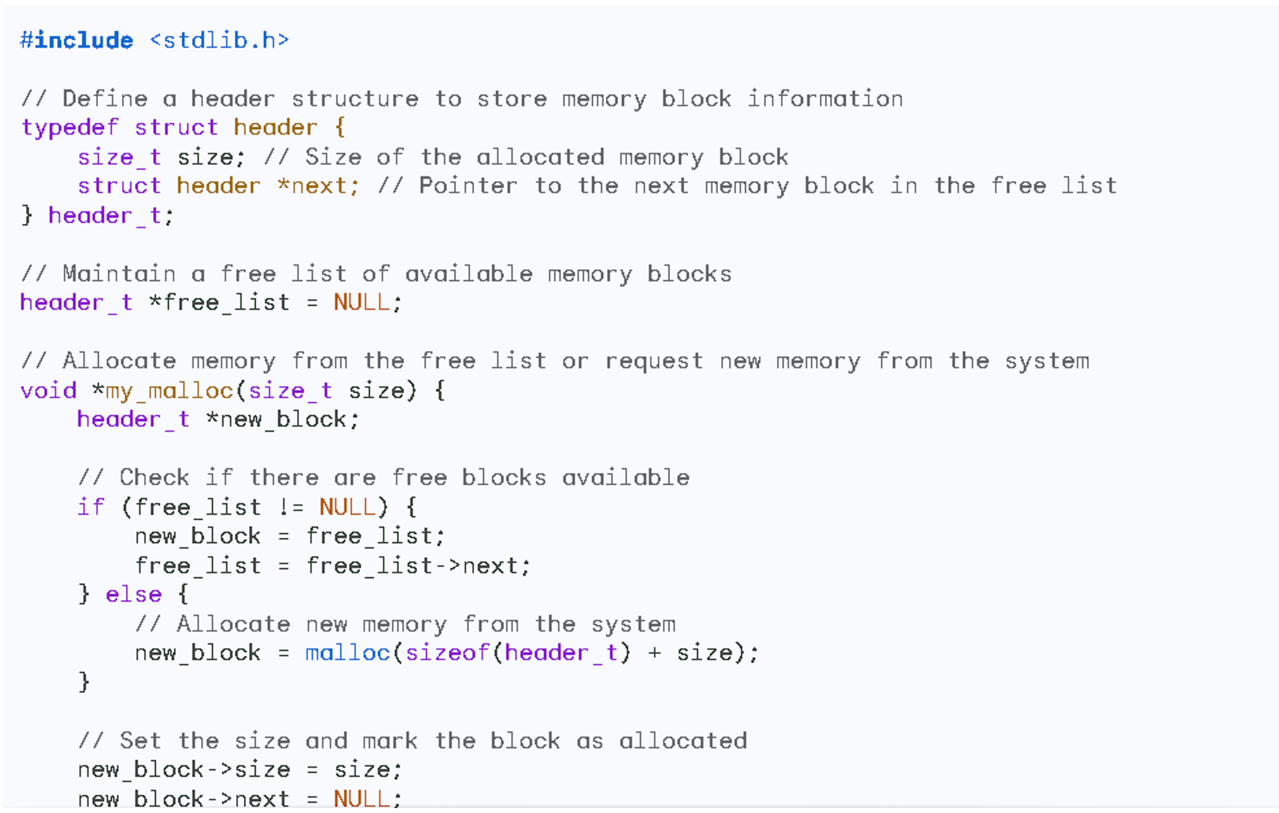
Static and Abstract Modifiers:

The "static" modifier is used to create class-level members that belong to the class rather than an instance. Static methods and variables can be accessed without creating an object of the class. The "abstract" modifier is used to define abstract classes and methods that serve as blueprints for derived classes. Abstract methods are defined without implementation in the abstract class.

Conclusion:

In summary, understanding and effectively using garbage collection, packaging, access modifiers, and static and abstract modifiers are fundamental skills for software developers:

Garbage collection ensures efficient memory management, reducing the risk of memory leaks and improving program stability. Packaging enhances code organization and promotes code reusability, making software development more manageable and collaborative. Access modifiers maintain the integrity of class members and support encapsulation and information hiding. Static and abstract modifiers allow developers to create class-level structures and design abstract classes that provide a blueprint for derived classes. By mastering these concepts, programmers can write more efficient, maintainable, and secure code, contributing to the success of software projects.

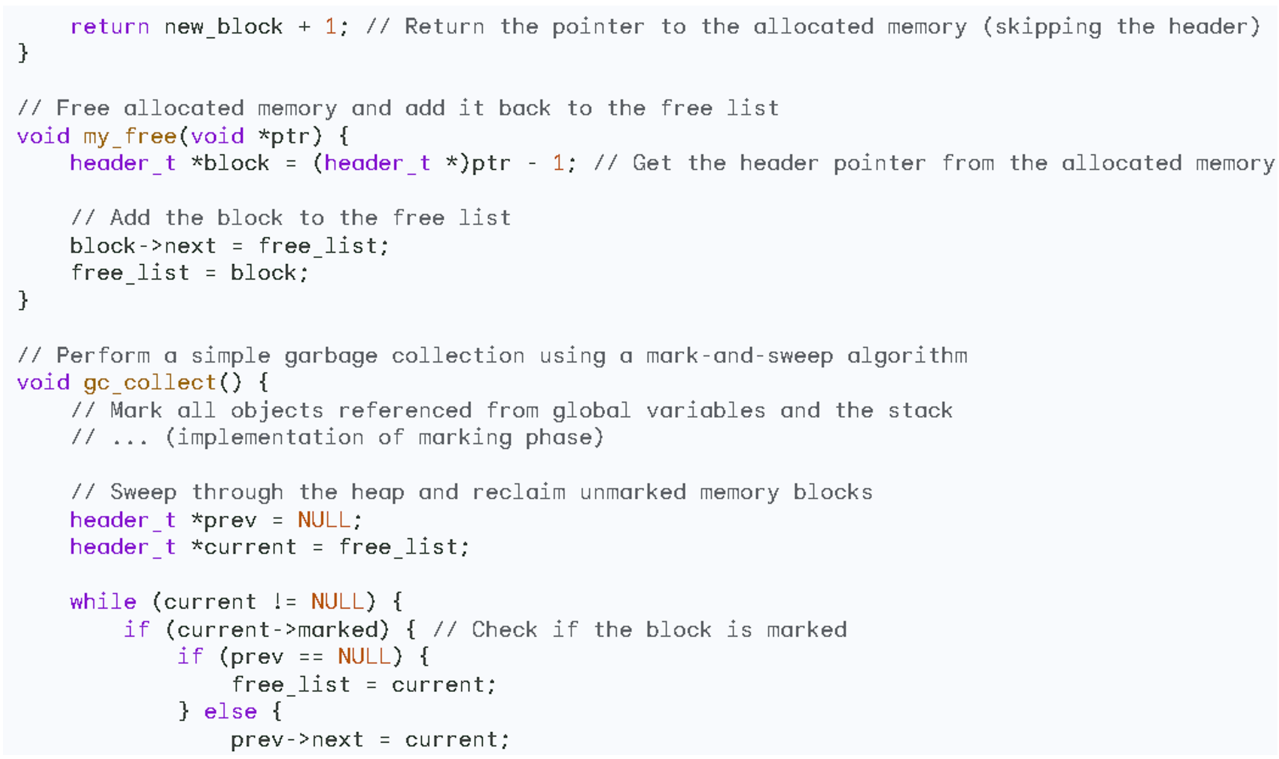


```c
#include <stdlib.h>

// Define a header structure to store memory block information
typedef struct header {
    size_t size; // Size of the allocated memory block
    struct header *next; // Pointer to the next memory block in the free list
} header_t;

// Maintain a free list of available memory blocks
header_t *free_list = NULL;

// Allocate memory from the free list or request new memory from the system
void *my_malloc(size_t size) {
    header_t *new_block;

    // Check if there are free blocks available
    if (free_list != NULL) {
        new_block = free_list;
        free_list = free_list->next;
    } else {
        // Allocate new memory from the system
        new_block = malloc(sizeof(header_t) + size);
    }

    // Set the size and mark the block as allocated
    new_block->size = size;
    new_block->next = NULL;
```

Figure 1: Enter Caption

program:

out put:



```c
    return new_block + 1; // Return the pointer to the allocated memory (skipping the header)
}

// Free allocated memory and add it back to the free list
void my_free(void *ptr) {
    header_t *block = (header_t *)ptr - 1; // Get the header pointer from the allocated memory

    // Add the block to the free list
    block->next = free_list;
    free_list = block;
}

// Perform a simple garbage collection using a mark-and-sweep algorithm
void gc_collect() {
    // Mark all objects referenced from global variables and the stack
    // ... (implementation of marking phase)

    // Sweep through the heap and reclaim unmarked memory blocks
    header_t *prev = NULL;
    header_t *current = free_list;

    while (current != NULL) {
        if (current->marked) { // Check if the block is marked
            if (prev == NULL) {
                free_list = current;
            } else {
                prev->next = current;
```

Figure 2: Enter Caption