Graphic Era deemed to be University DEHRADUN

PROJECT AND TEAM INFORMATION

Project Title

DYNAMIC MEOMORY ALLOCATOR(HEAP MANAGEMENT SYSTEM)

Student/Team Information

Team Name: Team #	Memory Minds DAA-IV-T032
Team member 1 (Team Lead)	Gupta,Riya-23022643 23022643@geu.ac.in 
Team member 2	Dwivedi,Shubh-230112175 230112175@geu.ac.in 

Team member 3	Kumari,Bhumi-23022707 23022707@geu.ac.in 

PROJECT PROGRESS DESCRIPTION

Project Abstract

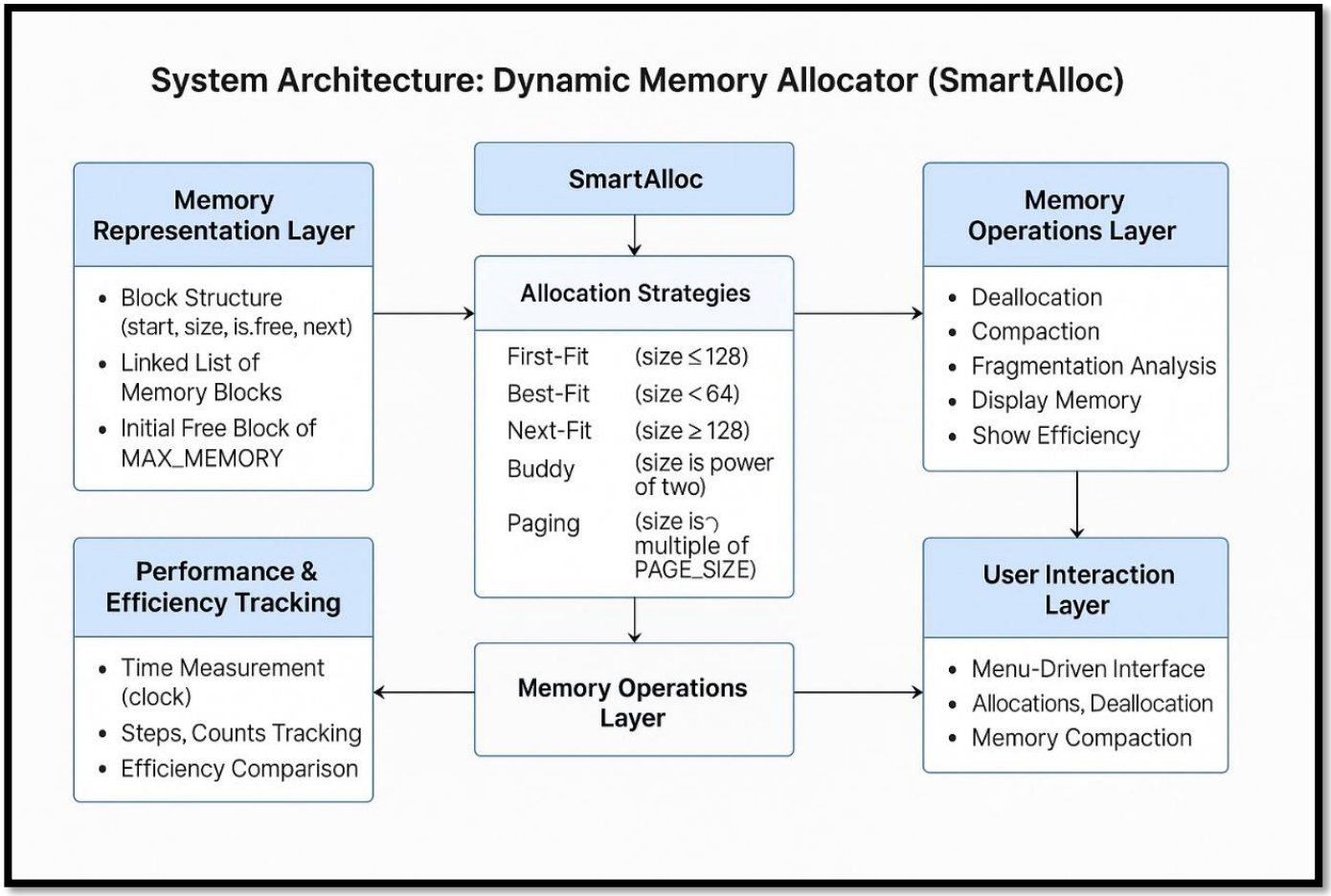
SmartAlloc is a comprehensive simulator for dynamic memory management in C. It supports multiple allocation strategies including First-Fit, Best-Fit, Next-Fit, Buddy Allocation, Paging, and a Smart Allocation mode that selects the most optimal strategy based on input size. This flexible system closely mimics real-world memory allocation techniques used in operating systems.

Users can interact through a terminal-based menu, allocate memory, deallocate memory using start addresses, perform fragmentation analysis, and compact memory. The system also calculates average time and steps taken by different strategies, helping evaluate algorithmic efficiency. SmartAlloc is both a learning tool and a testing ground for memory allocation mechanisms.

Updated Project Approach and Architecture

The system architecture is based on a linked list where each node (or Block) represents a chunk of memory with metadata such as size, start address, and status (free/used). The system uses malloc to simulate internal memory partitioning and tracks all operations through modular functions. Each strategy has a dedicated function while the “Smart Allocation” method dynamically chooses the best approach depending on the size.

Time and step tracking are implemented using clock() to measure allocation performance. The menu allows users to interactively run simulations, view memory layout, deallocate specific memory blocks, and analyze fragmentation. The design emphasizes modularity, efficiency tracking, and adaptability to make the system both educational and robust.


System Architecture: Dynamic Memory Allocator (SmartAlloc)
SmartAlloc
Memory Representation Layer
• Block Structure (start, size, is.free, next)
• Linked List of Memory Blocks
• Initial Free Block of MAX_MEMORY
Allocation Strategies
First-Fit (size ≤ 128)
Best-Fit (size < 64)
Next-Fit (size ≥ 128)
Buddy (size is power of two)
Paging (size is multiple of PAGE_SIZE)
Memory Operations Layer
• Deallocation
• Compaction
• Fragmentation Analysis
• Display Memory
• Show Efficiency
Performance & Efficiency Tracking
• Time Measurement (clock)
• Steps, Counts Tracking
• Efficiency Comparison
Memory Operations Layer
User Interaction Layer
• Menu-Driven Interface
• Allocations, Deallocation
• Memory Compaction

Tasks Completed

Task Completed	Team Member
Implemented First-Fit, Best-Fit and Next-Fit allocation strategies	Riya Gupta
Designed and implemented Smart Allocation Strategy selector	Riya Gupta
Developed Buddy Allocation with recursive splitting and power-of-two enforcement	Shubh Dwivedi
Implemented Paging Allocation using fixed-size 64-byte page simulation	Shubh Dwivedi
Implemented deallocation with merging of adjacent free blocks	Bhumi Kumari
Developed memory compaction(defragmentation)module	Bhumi Kumari
Created fragmentation analysis and efficiency summary reporting	All members collaborately

Challenges/Roadblocks

One challenge we faced was ensuring memory block integrity when splitting or merging during allocation and deallocation. Incorrect pointer updates led to segmentation faults, which were resolved through careful debugging of block links.

Managing the recursive buddy splitting was tricky, particularly when handling uneven block divisions. Ensuring all buddies retained the required structure and size took significant testing and verification.

Another issue was implementing an intelligent strategy selector. Designing the logic to switch between paging, buddy, and fit-based methods based on input size had to be carefully optimized for correctness and speed.

Input handling in the menu system also posed difficulties. Users could enter incorrect types or values, which we overcame by validating inputs before processing choices.

Efficient tracking of time and steps for each algorithm required careful use of the clock() function. Averaging metrics across multiple operations helped in eliminating noisy readings and provided meaningful results.

When analyzing fragmentation, distinguishing between internal (within used blocks) and external (between free blocks) fragmentation became complicated after compaction. We resolved this by iterating carefully through the list and classifying each block properly.

Finally, integrating compaction with active memory tracking required updates to block addresses and the insertion of a new free block without disrupting the list. The structure was maintained through pointer reassignments and block reallocation.

Tasks Pending

Task Pending	Team Member (to complete the task)
Implemented memory state export/import feature.	Shubh Dwivedi
Enhance user input validation and error messaging.	Bhumi Kumari
Add visual memory map using ASCII formatting.	Riya Gupta
Create allocation history and usage log.	Riya Gupta
Improve efficiency tracking to support logging.	Shubh Dwivedi
Finalize test case coverage and boundary conditions.	Bhumi Kumari
Prepare demo presentation and performance charts.	Entire team

Project Outcome/Deliverables

The project delivers a working, interactive dynamic memory management simulator in C. It includes six allocation strategies (First-Fit, Best-Fit, Next-Fit, Buddy System, Paging, and Smart Allocation), performance tracking (time and steps), memory deallocation, compaction, and fragmentation analysis.

Planned optional deliverables include a memory usage visualization, import/export of memory states, and an allocation log. These features aim to enhance usability and traceability. The tool serves both academic and performance evaluation purposes for understanding OS-level memory allocation strategies.

Progress Overview

Approximately 85% of the project has been completed. All core allocation strategies are functional and tested. Deallocation, compaction, fragmentation analysis, and the Smart Allocator have been successfully implemented.

Pending tasks mostly involve enhancements such as logging, visualization, and user input refinement. The overall structure is complete, and testing is ongoing. We are on track to finalize and polish the project before the submission deadline.

Codebase Information

The code is maintained in a single C source file containing all modules. Functions are grouped by strategy and structured clearly with descriptive naming. Efficiency tracking variables are global and reset between runs.

Key features include smart strategy selection, performance tracking using clock(), and linked list management of memory blocks. A menu-driven interface handles all interaction. Code comments and debug print statements are included for clarity and testing.

GitHub Link-
https://github.com/shubh9125/Dynamic_Memory_Allocator_-Heap_Management_System-

Testing and Validation Status

Test Type	Status (Pass/Fail)	Notes
First-Fit, Best-Fit,Next-fit	Pass	All algorithms correctly allocates and split blocks
Buddy Allocation test	Pass	Recursive splitting works with power-of-two validation
Paging Allocation	Pass	Allocates multiple 64-bytes pages as expected
Deallocation and merging	Pass	Merges adjacent free blocks correctly
Compaction test	Pass	Shifts used blocks and resets start addressess effectively.
Fragmentation analysis	Pass	Accurately reports internal and external fragmentation.
Smart alloaction decision	Pass	Correctly selects Best-Fit strategy based on size.

Deliverables Progress

All core deliverables have been completed, including five allocation strategies, deallocation, compaction, fragmentation analysis, and smart strategy selection. The CLI interface is responsive and well-tested.

Pending deliverables like memory visualization, logging, and memory state import/export are under active development. These enhancements will make the tool more informative and user-friendly for demonstration purposes.