**Computer System (CS527)**

**[DYNAMIC MEMORY ALLOCATION]**

**[Project]**

**A Project Report Submitted by**

**SAMPANNA SINGH**

**AARIF**

**GULAM MUSTAFA**

**MAHAVIR PRASAD SAH**

**Master of Technology**

**in**

**Computer Engineering (AI)**



**Department of CSE,**

**Indian Institute of Technology, Rupnagar**

**2023-24**

Undertaking about Originality of Work

We hereby certify that we are the sole authors of this Project report and that neither any part of this Project report nor the whole of the Project has been submitted for a project report by other student(s) to any other University or Institution.

We certify that, to the best of our knowledge, the current Project report does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations or any other material from the work of other people included in our Project report, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that we have included copyrighted material that surpasses the boundary of fair dealing within the meaning of the Indian Copyright (Amendment) Act 2012, we certify that we have obtained a written permission from the copyright owner(s) to include such material(s) in the current IDP/UDP Project report and have included copies of such copyright clearances to our appendix.

We have checked the write up of the present Project report using anti-plagiarism database and it is in the allowable limit. In case of any complaints pertaining to plagiarism, we certify that we shall be solely responsible for the same and we understand that as per norms, University can even revoke M.Tech degree conferred upon the student(s) submitting this Project report, in case it is found to be plagiarised.

**Team:**

|  |  |
| --- | --- |
| **Entry Number** | **Name** |
| 2023AIM1017 | Sampanna Singh |
| 2023CSZ0017 | AARIF |
| 2023CSM1004 | GULAM MUSTAFA |
| 2023CSM1024 | MAHAVIR PRASAD SAH |

Acknowledgments

I extend my sincere gratitude to Dr. Neeraj Goel, Tutor of Computer System, CSE Department, Indian Institute of Technology, for providing an opportunity to work under his kind control. I am indebted to the authorities & staff of CE Department, Indian Institute of Technology, Rupnagar for their generous help at each step of formulation and completion of this work.

I would like to thank to the Teaching Assistant for helping me to have a complete detailed study of the Operating System, Computer System in detail. It was due to their constant guidance and supervision that I was able to understand most of the concepts thoroughly.

Finally, I also extend my thanks and gratitude to my other batch mates, students of CSE, Semester I for their cooperation and for their endless help.

Index

Abstract V

List of Table VI

List of Figures VI

1 Chapter 1:Introduction 1-7

1.1 Computer System and DMA 1

1.2 Problem Definition 2

1.3 Motivation 3-4

1.4 Scope 4-5

1.5 Project Objective 6

1.6 Project Management 7

1.6.1 Gantt Chart 7

1.6.2 Timeline……………………………………………………………….…7

2 Chapter 2: Basic Of DMA 8-12

2.1 Introduction 8-9

2.2 Implict list 9-11

2.3 Explict List 11-12

3 Chapter 3 : Project - Memory Allocation Simulation 13-

3.1 Project Statement 13

3.2 Features 14-16

3.3 Functional Specification 16-17

3.4 Code

3.5 Visualization

Conclusion

References

Abstract

The Memory Allocation Simulation, employing HTML, CSS, and JavaScript, is an educational web-based tool designed to explore memory allocation strategies. Users interact via controls to initialize a memory heap, employ First Fit, Next Fit, or Best Fit strategies, and deallocate memory blocks for reuse. Through an intuitive interface, users set heap sizes, allocate memory blocks, and view real-time visualizations displaying block details like addresses, sizes, and allocation statuses. Aimed at enhancing understanding, the tool provides hands-on learning, validated inputs, and feedback messages for successful or unsuccessful operations. Geared toward students and enthusiasts in computer science, this resource illuminates the practical implications of memory allocation in software development.

Using HTML, CSS, and JavaScript, the Memory Allocation Simulation facilitates hands-on exploration of memory allocation strategies. It allows users to initialize heap sizes, allocate memory using different strategies, and view visual representations of memory blocks. With user-friendly controls and informative feedback, the simulation guides users through valid inputs and illustrates outcomes. This educational tool is tailored for individuals seeking practical insights into memory management within computer system and Architecture.

List of Tables

Table 1 1 Timeline of project………………………………………….……………….….7

List of Figures

FIgure1 2 Gantt’s chart………………………………………….……………….….7

**1. Introduction**

**1.1 Computer System**

A computer system is a sophisticated and interconnected assembly of hardware, software, and peripherals designed to execute instructions, process data, and perform various tasks. It comprises several essential components working together to facilitate computing operations and fulfill user requirements**.**

**Key Components of a Computer System:**

* Central Processing Unit (CPU), Memory (RAM, ROM, Storage Devices),Input and Output Devices, O.S, Software Application, Networking Component.

**Functionality of a Computer System:**

A computer system functions by following a series of steps known as the information processing cycle. This cycle includes inputting data, processing that data using the CPU, storing results in memory or storage, outputting the processed information, and finally, providing mechanisms for user interaction and feedback.

**Evolution and Impact:**

Computer systems have evolved significantly over time, becoming smaller, faster, and more powerful. They have revolutionized numerous aspects of society, impacting fields such as education, healthcare, business, communication, entertainment, scientific research, and many more.

In essence, a computer system serves as a versatile and indispensable tool, capable of executing diverse tasks by processing data, performing computations, and enabling interaction between humans and machines.

**1.2 Problem Definition**

Develop and implement an efficient dynamic memory allocation system that Visualize memory usage and allocation in a computer system for managing varying memory requirements of programs during runtime.

**Dynamic Memory Management System Design:** Design and develop algorithms, data structures, and mechanisms to efficiently allocate and deallocate memory dynamically. Create an effective memory management system that minimizes fragmentation and maximizes memory utilization.

**Memory Allocation Algorithms:** Implement and evaluate various memory allocation algorithms such as First Fit, Best Fit, Next Fit to determine the most suitable method for allocating memory blocks dynamically. Analyze their performance in terms of speed, fragmentation, and overall efficiency.

**Memory Leak Detection and Prevention:** Develop mechanisms to detect and prevent memory leaks, ensuring that allocated memory is properly deallocated when no longer needed. Implement checks and balances to identify and handle memory leaks during runtime.

**Optimization and Efficiency:** Optimize the dynamic memory allocation system to minimize overheads and improve performance. Implement strategies to reduce memory fragmentation and enhance allocation speed without compromising reliability.

**Error Handling and Stability:** Incorporate error handling mechanisms to maintain system stability and prevent issues such as segmentation faults or memory corruption. Ensure robust error detection and recovery strategies.

**1.3 Motivation:**

Here are several motivations for exploring and improving dynamic memory allocation:

**Optimized Resource Utilization:** Efficient memory management is crucial for maximizing the utilization of available resources in a computer system. Dynamic memory allocation allows for better utilization by allocating memory on-demand and releasing it when no longer needed, reducing wastage.

**Support for Dynamic Data Structures:** Dynamic memory allocation enables the creation and management of dynamic data structures such as linked lists, trees, and resizable arrays. These data structures can adapt their size during program execution, accommodating varying data needs and enhancing program flexibility.

**Adaptability to Varying Program Requirements:** Programs often have changing memory requirements during execution. Dynamic memory allocation caters to these varying needs, enabling programs to request and release memory as necessary, promoting efficient usage and responsiveness.

**Enhanced Program Flexibility:** Dynamic memory allocation allows for the creation of applications that can handle different data sizes and types dynamically. It offers flexibility for programs to adjust memory allocations based on user inputs or runtime conditions.

**Reduced Fragmentation:** Efficient dynamic memory allocation algorithms aim to minimize memory fragmentation. By allocating and deallocating memory blocks dynamically, fragmentation issues can be reduced, optimizing memory usage and avoiding performance degradation.

**Memory Leak Detection and Prevention:** Implementing dynamic memory allocation systems involves managing memory effectively, which includes detecting and preventing memory leaks. Proper handling of memory allocation and deallocation helps in preventing memory leaks, ensuring better program stability and reliability.

**Support for Modern Computing Environments:** In modern computing, where applications often run concurrently or on resource-constrained devices, efficient memory management through dynamic allocation becomes even more critical to meet performance and resource optimization goals.

**1.4 Scope:**

The scope for dynamic memory allocation is vast and encompasses various aspects within the realm of computer science and software development. Here are some key areas that highlight the scope and importance of dynamic memory allocation:

**Programming Languages and Software Development:** Dynamic memory allocation plays a significant role in programming languages like C, C++, Java, Python, and others. Understanding and implementing efficient memory allocation strategies are crucial for software developers to create optimized and scalable applications.

**Operating Systems:** Dynamic memory allocation is fundamental in operating systems for managing memory resources efficiently. Memory management techniques, including dynamic allocation, are critical components in ensuring stable and reliable system performance.

**Embedded Systems and IoT:** In resource-constrained environments such as embedded systems and IoT (Internet of Things) devices, efficient memory usage is vital. Dynamic memory allocation helps in optimizing memory utilization while addressing the limitations of hardware resources.

**System Design and Architecture:** Architects and designers focus on efficient memory management to create high-performance computing systems. Strategies related to dynamic memory allocation impact system design decisions and influence the overall system architecture.

**Real-time Systems and Performance-critical Applications:** In real-time systems and performance-critical applications (such as gaming, multimedia processing, and scientific simulations), efficient memory allocation directly impacts system responsiveness and performance.

**Database Systems and Big Data Processing:** Memory allocation strategies are crucial in database systems and big data processing frameworks. Effective memory management enhances data processing speed and scalability in these systems.

**Security and Reliability:** Memory management vulnerabilities, including memory leaks and buffer overflows, can pose security risks. Understanding dynamic memory allocation helps in building secure and reliable software by mitigating such vulnerabilities.

**Research and Optimization Techniques:** Ongoing research focuses on developing improved memory allocation algorithms, fragmentation reduction methods, and optimizing memory usage to address evolving computing requirements.

**1.5 Project Objective:**

The primary objective of the Memory Allocation Simulation project is to develop an interactive and educational web-based tool that facilitates the exploration and understanding of various memory allocation strategies. By leveraging HTML, CSS, and JavaScript, the project aims to offer users, particularly students and enthusiasts in computer science, a practical platform to interactively experiment with different memory allocation methods.

The project seeks to provide users with hands-on experience in memory management concepts by allowing them to initialize memory heaps, allocate memory blocks using different strategies such as First Fit, Next Fit, and Best Fit, and observe real-time visualizations illustrating the allocation and status of memory blocks. The main goal is to bridge the gap between theoretical knowledge and practical application, enabling users to comprehend the implications and effectiveness of different memory allocation techniques within the realm of software development.

Overall, the objective is to create an accessible and user-friendly resource that fosters a deeper understanding of memory management principles, equipping users with practical insights into the significance of efficient memory allocation in designing and optimizing software systems.

**1.6 Project Management:**

**1.6.1 Gantt’s chart:**

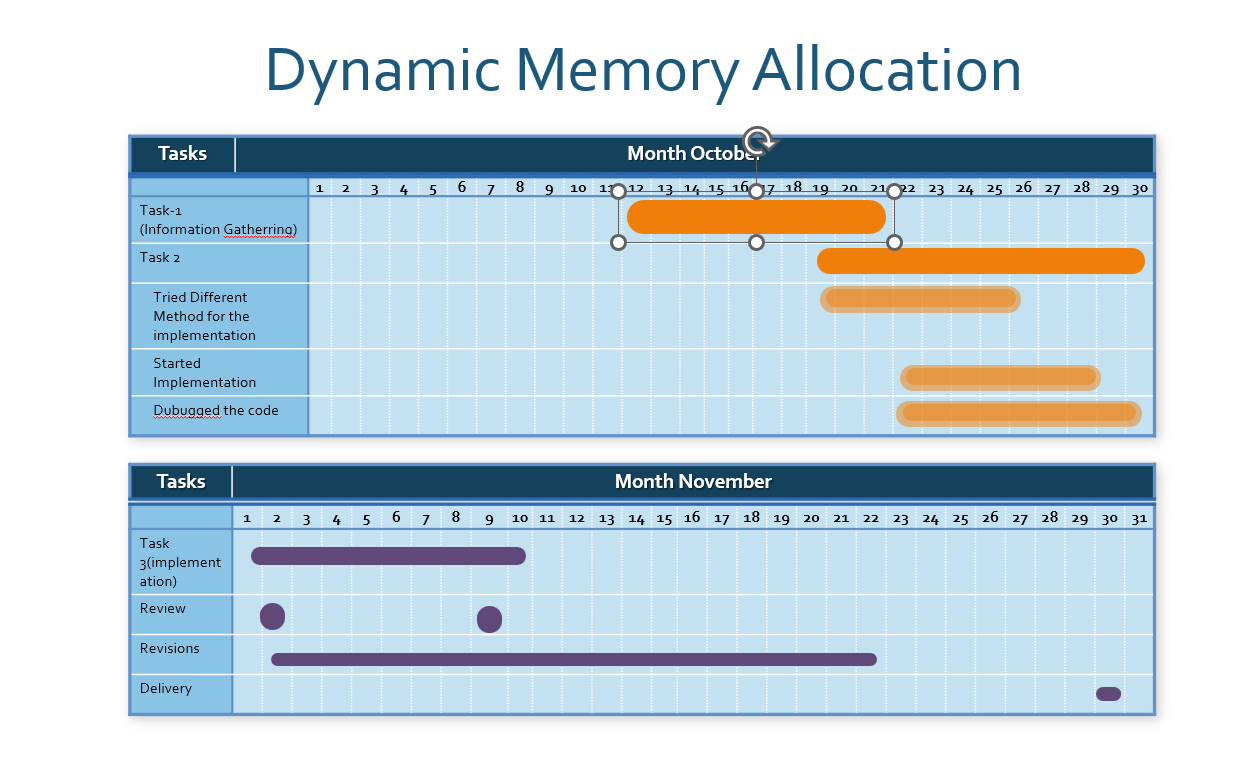


Figure 1 Gantt's chart

**1.6.2 Timeline:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Start Date** | **End Date** | **Duration** |
| **Information Gathering** | Oct 12, 2023 | Oct 21, 2023 | 11 days |
| **Explored Different Technique** | Oct 19, 2023 | 0ct 25, 2023 | 11 days |
| **Started Implementation** | Oct 22, 2023 | Oct 30, 2023 | 8 days |
| **Debugging** | Oct 22, 2023 | Oct 30, 2023 | 8 days |
| **Review** | Oct 2, 2023 | Oct 9 | 2 days |

**2: Basic of DMA**

**2.1 Introduction**

Dynamic memory allocation refers to the process by which a computer program requests memory space from the operating system during its execution, enabling the program to manage memory resources dynamically. Unlike static memory allocation, where memory is allocated at compile time and remains fixed throughout program execution, dynamic memory allocation allows programs to allocate memory as needed during runtime and release it when no longer required.

**On-Demand Memory Management:** Dynamic memory allocation allows programs to adapt to varying memory needs during execution. It enables the allocation and deallocation of memory blocks based on program requirements, allowing for flexibility in managing memory resources.

**Heap Allocation:** In many programming languages like C, C++, and others, dynamic memory allocation commonly involves allocating memory from the heap. The heap is a region of memory separate from the program's stack, providing a pool of memory that can be dynamically allocated and freed.

**Allocation Functions:** Programming languages provide functions or operators (e.g., malloc, calloc, realloc in C/C++) to request memory dynamically. These functions allocate a specified amount of memory and return a pointer to the allocated memory block.

**Deallocation:** Alongside allocation, proper deallocation of dynamically allocated memory is crucial to prevent memory leaks. Memory that is no longer needed should be explicitly released using corresponding deallocation functions (e.g., free in C/C++) to return the memory to the system for reuse.

**Dynamic Data Structures:** Dynamic memory allocation is instrumental in managing dynamic data structures such as linked lists, trees, and dynamic arrays. These structures can grow or shrink in size during program execution, requiring dynamic allocation and deallocation of memory.

**Memory Fragmentation:** Improper management of dynamic memory can lead to memory fragmentation, where the available memory becomes fragmented into small, unusable chunks, affecting the overall efficiency of memory utilization.

**Resource Management:** Effective use of dynamic memory allocation is essential for efficient resource management in programs, ensuring optimal utilization of memory resources and preventing unnecessary memory wastage.

Dynamic memory allocation plays a vital role in enabling programs to handle varying memory requirements and manage memory resources effectively during runtime, contributing to the flexibility and efficiency of computer programs.

* **Implicit List:**

An approach where the available blocks of memory within the heap are managed implicitly without an explicit data structure to track free blocks. Here's an explanation of the implicit list method in dynamic memory allocation:

Implicit List Method in Dynamic Memory Allocation:

**No Explicit Data Structure:** In the implicit list method, the management of free memory blocks doesn't rely on maintaining an explicit data structure (such as linked lists or tables) to track available memory blocks. Instead, free blocks are identified and managed based on characteristics embedded within the memory blocks themselves.

**Identification of Free Blocks:** Free memory blocks are identified implicitly through specific markers or attributes within the memory. These markers, often located in the block headers or footers, signify whether a particular block is free or allocated.

**Allocation and Deallocation:** When an allocation request is made, the memory allocator traverses the memory pool to locate a suitable free block that matches the requested size. Once identified, the allocator allocates the required portion from the block, potentially splitting the block and leaving the remaining segment as a smaller free block.

**Merging Free Blocks:** After deallocation, adjacent free blocks in the memory pool might be merged together to form larger contiguous free blocks. This process helps in reducing fragmentation and enabling the allocation of more extensive memory blocks when needed.

**Fragmentation Consideration:** The implicit list method aims to manage memory fragmentation, ensuring efficient use of available memory. However, it may still incur internal fragmentation when the allocated memory is slightly larger than necessary, leading to unused space within the block.

**Simplicity and Performance Trade-offs:** The implicit list method is relatively simple in its approach compared to explicit list-based methods. However, while it reduces the overhead of managing an explicit data structure, it may involve increased traversal or search time to find appropriate free blocks for allocation.

This method is utilized by many dynamic memory allocation systems to efficiently manage memory blocks without the need for extensive data structures, making it suitable for various computing environments.

In essence, the implicit list method in dynamic memory allocation involves managing available memory blocks implicitly, relying on markers within the memory blocks themselves to identify and handle free memory segments.

**2.****3 Explict list:**

An explicit free list is a method used to manage free memory blocks explicitly by maintaining a data structure that tracks available blocks of memory within the heap. This data structure keeps a record of free memory blocks, enabling efficient allocation and deallocation of memory.

Here's an overview of the explicit free list in dynamic memory allocation:

**Data Structure for Free Blocks:** Unlike implicit free lists that identify available memory blocks implicitly within the memory space, the explicit free list uses a dedicated data structure (such as a linked list, binary tree, or segregated lists) to maintain information about free memory blocks.

**Tracking Free Blocks:** The explicit free list keeps track of available memory blocks by storing pointers or references to these blocks in the data structure. Each entry in the list corresponds to a free memory block and contains information about its size, status, and possibly pointers to adjacent blocks.

**Allocation Process:** When a memory allocation request is made, the allocator searches the explicit free list data structure to find a suitable block that matches the requested size. Once a block is found, it is allocated to fulfill the request, potentially splitting the block if necessary and updating the list accordingly.

**Deallocation and Block Coalescing:** Upon deallocation, the freed memory block is marked as available. The allocator may perform block coalescing, merging adjacent free blocks to form larger contiguous blocks, reducing fragmentation and enhancing future allocation opportunities.

**Fragmentation Handling:** Explicit free lists aim to reduce both external and internal fragmentation. By keeping track of free blocks explicitly, the allocator can efficiently merge adjacent blocks, minimizing wasted space and improving memory utilization.

**Complexity and Overhead:** While explicit free lists provide efficient management of free blocks, they require additional memory overhead to maintain the data structure. Operations such as insertion, deletion, and traversal within the list might also introduce computational overhead.

**Performance Considerations:** The explicit free list method can facilitate faster block allocation compared to implicit methods, as it directly accesses and manipulates a structured list of free blocks. However, the efficiency may vary based on the chosen data structure and allocation patterns.

Overall, the explicit free list method in dynamic memory allocation utilizes a dedicated data structure to manage available memory blocks, offering more direct control and efficient handling of free memory segments within the heap.

**3.** **Memory Allocation Simulation**

**3.1 Project Statement:**

The objective of this project is to create a web-based simulation tool for memory allocation strategies. The tool will allow users to interactively visualize and understand how memory allocation techniques like First Fit, Next Fit, and Best Fit work within a simulated memory environment.

The project involves the development of an interactive web application that simulates a memory management system. The system will demonstrate the allocation and deallocation of memory blocks using different strategies, providing users with a hands-on experience to comprehend the behavior and effectiveness of various allocation methods.

**5.2 Features:**

Here are the features of the Memory Allocation Simulation project:

1. Interactive Controls:
   * Initialization of Heap: Users can set the initial size of the memory heap.
   * Allocation Strategies: Support for First Fit, Next Fit, and Best Fit algorithms for memory block allocation.
   * Deallocation: Capability to release allocated memory blocks for reuse.
2. Visualization:
   * Memory Block Representation: Visual display of memory blocks showing allocated and free segments.
   * Address and Size Display: Clear visualization indicating block addresses, sizes, and allocation statuses.
   * Real-time Updates: Instantaneous visualization updates reflecting allocation and deallocation actions.
3. Allocation Strategies:
   * First Fit: Allocate the first available memory block that is large enough to hold the requested size.
   * Next Fit: Allocate the next available memory block from the last allocation point that satisfies the size requirement.
   * Best Fit: Allocate the smallest available memory block that fits the requested size most closely.
4. User Interaction:
   * User Input Validation: Ensure valid input for memory size, block sizes, and pointer names.
   * Error Handling: Clear feedback on invalid inputs, failed allocations due to insufficient space, or already allocated pointer names.
5. Dynamic Updates:
   * Real-time Memory Table: Display of allocated blocks in a tabular format with addresses, sizes, and allocation statuses.
   * Visualization Adjustment: Dynamically adjust the visual representation based on allocation and deallocation actions.
6. Educational Purpose:
   * Learning Tool: Aimed at students and enthusiasts to understand memory allocation strategies.
   * Practical Experimentation: Facilitate hands-on learning of memory management concepts through interactive experimentation.
7. Technological Stack:
   * Utilization of HTML, CSS, and JavaScript: Development using web technologies for an interactive and accessible platform.
8. User-Friendly Interface:
   * Intuitive Design: A clean and easy-to-navigate interface for seamless user interaction.
   * Clear Feedback: Informative messages and visual cues for successful and unsuccessful memory operations.
9. Simulation Environment:
   * Simulated Memory Heap: Mimicking memory management operations to simulate real-world scenarios.
   * Reusability: Deallocated memory blocks become available for subsequent allocations, mimicking real memory behavior.

10.Comprehensive Understanding:

* + Facilitate Experimentation: Allow users to compare and contrast different allocation strategies for a comprehensive understanding of their strengths and limitations.

5.3 Functional Specification:

The functional specifications for the Memory Allocation Simulation are as follows:

**1.Initialization Functionality**:

* **Heap Initialization:** Users can set the initial size of the memory heap using an input field.
* **Validation:** Validates the input for heap size, ensuring it is a positive integer greater than zero.
* **Initialization Action**: Initializes the memory heap upon user action, creating the initial block with a specified size and 'brk\_ptr' label.

**2.Memory Allocation Functions:**

* **First Fit Allocation:**
* Users can allocate memory blocks using the First Fit strategy by specifying block size and pointer names.
* Validates input for block size and pointer name, ensuring they meet the criteria.
* Allocates memory based on the First Fit algorithm:
* Finds the first available block that can accommodate the requested size.
* Updates the memory table and visualization accordingly.
* Provides feedback on successful or failed allocations.
* **Next Fit Allocation:**
* Allows users to allocate memory blocks using the Next Fit strategy by specifying block size and pointer names.
* Validates input for block size and pointer name, ensuring they meet the criteria.
* Allocates memory based on the Next Fit algorithm:
* Begins the search for the next available block from the last allocated index.
* Updates the memory table and visualization accordingly.
* Provides feedback on successful or failed allocations.
* **Best Fit Allocation:**
* Enables users to allocate memory blocks using the Best Fit strategy by specifying block size and pointer names.
* Validates input for block size and pointer name, ensuring they meet the criteria.
* Allocates memory based on the Best Fit algorithm:
* Finds the smallest available block that fits the requested size most closely.
* Updates the memory table and visualization accordingly.
* Provides feedback on successful or failed allocations.

**3.Deallocation Functionality:**

* **Memory Deallocation:**
* Allows users to deallocate memory blocks by specifying the pointer name associated with the allocated block.
* Validates the input for the pointer name, ensuring it corresponds to an allocated block.
* Deallocates the memory block associated with the specified pointer:
* Frees up the memory block for subsequent allocations.
* Coalesces adjacent free memory blocks if applicable.
* Updates the memory table and visualization accordingly.
* Provides feedback on successful or failed deallocations.

**4.Memory Visualization:**

* **Memory Table Display:**
* Maintains a table displaying the memory blocks with their addresses, sizes, and allocation statuses (free or allocated).
* Updates the table in real-time after allocation or deallocation actions.
* **Visual Representation:**
* Provides a visual representation of memory blocks in a graphical format.
* Displays allocated and free memory segments in a simulated heap.
* Dynamically adjusts visualization based on allocation and deallocation actions.

**5.User Interface and Feedback:**

* **Input Validation Feedback:**
* Provides clear and concise messages for invalid input entries, guiding users on the required input format.
* **Operation Feedback:**
* Offers informative messages to users indicating the success or failure of memory allocation and deallocation operations.
* **Interactive Controls:**
* Provides user-friendly controls such as buttons and input fields for seamless interaction.

**6.Error Handling:**

* **Error Prevention:**
* Prevents invalid memory operations by thoroughly validating user inputs and blocking actions that violate specified constraints.
* **Error Messaging:**
* Prompts informative error messages to guide users in rectifying input or understanding the reason for failed operations.

These functional specifications outline the various functionalities and behaviors implemented within the Memory Allocation Simulation, allowing users to interact with and comprehend memory allocation strategies effectively.

**5.4 Code:**

**5.5 Simulation:**

**Conclusion:**

The Memory Allocation Simulation stands as an impactful and user-centric educational resource designed to elucidate the intricacies of memory management strategies. By harnessing the capabilities of HTML, CSS, and JavaScript, this interactive web-based tool empowers users, particularly students and enthusiasts in computer science, to delve into the practicalities of memory allocation.

Through a straightforward and intuitive interface, users engage in initializing memory heaps, experimenting with diverse allocation strategies like First Fit, Next Fit, and Best Fit, and observing real-time visualizations illustrating block details. This simulation fosters a deeper comprehension of memory management principles, bolstered by error-preventing input validations and informative feedback messages that accompany each memory operation.

Ultimately, the Memory Allocation Simulation serves as a bridge between theoretical knowledge and practical application, offering an accessible and engaging platform for individuals to grasp the nuances of memory allocation within the context of software development. As an educational tool, it equips users with valuable insights, enabling them to appreciate the significance of efficient memory management in the creation and optimization of software systems.

**References**

* **Class presentation note.**
* <http://courses.washington.edu/css342/zander/css332/memoryoop.html>
* <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9847757>
* <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6910522>
* <https://en.wikipedia.org/wiki/Memory_management>
* http://www.cs.ecu.edu/karl/3300/spr14/Notes/C/Memory/heap.html
* <https://www.youtube.com/watch?v=I9Qxm2zYZv8&list=PLyboo2CCDSWnhzzzzDQ3OBPrRiIjl-aIE&index=24>