GIK Institute of Eng. and Tech.

Faculty of computer Sciences and Engineering

Data Structures and Algorithms

Assignment # 2

Total Marks: **100**

Due: Thursday, October 23, 2025

You are free to consult each other for verbal help. However, **copying or sharing the code with each other will not only result into the cancellation of the current assignment, it may impact your grade in all the assignments and exams as well**.

Assignment

Title: Memory Allocation and Management using Dynamic Super-Blocks in C++
Objective:
The goal of this Assignment is to design and implement a memory management system using dynamic memory allocation in C++. The memory will be managed in fixed-sized building blocks, and the system will allow users to allocate memory for variable-sized super-blocks and deallocate memory either entirely or partially. The project will require maintaining a linked list of super-blocks, each holding a user-defined string, and implementing a variety of functions for memory management.

Problem Overview:
You are tasked with creating a memory management system in C++ that operates on a pool of fixed-sized memory blocks. The system will manage memory by creating super-blocks—variable-sized blocks composed of contiguous memory units. The project will include functions for allocating memory, deallocating either part or entire super-blocks, and maintaining a linked list to manage the blocks. Additionally, users will be able to input strings that will be stored in super-blocks, and you will implement functionality to display the memory pool and linked list contents.

Tasks and Function Descriptions:
1. Define SuperBlock Structure:
You will first define a structure to represent a SuperBlock. Each super-block will be stored as a node in a linked list, with each block storing:

- BlockId: Unique ID of the block.
- startIndex: Starting index of the block in the memory pool.
- size: Size of the memory block.
- data: The actual data (string) stored in the block.
- next: A pointer to the next super-block in the list.

2. Memory Pool Initialization:
Implement a function to initialize the memory pool. The memory pool is an array of char where each building block represents one character. At the beginning, all blocks will be marked as 'E' for empty.
char memoryPool[64]; // Memory pool of 64 building blocks.

void initializeMemoryPool();

- **Description**: This function will initialize all elements of memoryPool to 'E' (empty), representing available memory blocks.

4. Display the Memory Pool
Write a function to display Memory Pool.

void displayMemoryPool();

- **Description**: Displays the current status of the memory pool, showing which blocks are occupied and which are free.

3. Find Available Memory Block:
Write a function to find the first available contiguous set of memory blocks that can accommodate a super-block of a given size.
int findAvailableBlock(int size);

- **Description**: Finds the first contiguous sequence of empty blocks that can accommodate a new allocation of the given size. Returns the starting index of the first available block, or -1 if no suitable block is found.

4. Append SuperBlock to Linked List:
Implement a function to append a new SuperBlock to the linked list after finding the appropriate available memory in the pool.

SuperBlock* Append(int startIndex, int size);

- **Description**: The function will create a new SuperBlock with the given start index and size, then append it to the linked list of super-blocks. It returns a pointer to the new super-block.

5. Allocate SuperBlock for String:
Implement a function to allocate a super-block of appropriate size to store a user input string. This function will allocate memory in the pool, mark the blocks as occupied, store the string in the super-block, and append it to the linked list.

SuperBlock* allocateSuperBlockForString(const std::string &str);

- **Description**: This function should take a string as input, determine how many building blocks are needed, and allocate memory accordingly. It will also store the string in the super-block and update the linked list. The function first finds available space, stores the string in the super-block and the memory pool, and returns a pointer to the newly allocated super-block.

6. Display the Linked List and Memory Pool:
Create a function to display the contents of the linked list and memory pool. Each memory block in the pool should be displayed with its contents and each super-block's information should be displayed from the linked list.

void DisplayLinkedList();

- **Description**: Displays the linked list of allocated super-blocks, showing each block's BlockId, its data, the start and end indices in the memory pool, and the size of the block.

7. Deallocate Entire SuperBlock:
Write a function to deallocate a super-block by marking all its blocks as empty in the memory pool and removing it from the linked list.

void deallocateSuperBlock(int BlockId);

- **Description**: This function dallocates the memory block with the specified BlockId. It will search for a super-block in the linked list using ***BlockId***, and find/read the corresponding ***startIndex*** of the block to be deleted. It marks the corresponding memory pool blocks as free ('_'), and removes/deletes it from the linked list.

8. Deallocate Part of a SuperBlock:
Implement a function to deallocate part of a super-block, freeing a specified number of contiguous blocks within the super-block, starting from start of the block.

void deallocatePartOfSuperBlock(int BlockId, int partSize);

- **Description**: This function deallocates a specified portion of a super-block in a memory pool based on a given ***BlockId*** and ***partSize***. It first finds the super-block by traversing a linked list using ***BlockId***, and find/read the corresponding ***startIndex*** of the block to be deleted. If the ***partSize*** exceeds or matches the super-block's size, it returns an error.
  Otherwise, it marks the specified portion of the super-block as free ('_') in the memory pool and adjusts the super-block's starting index and size. If no matching ***BlockId*** is found, an error message is displayed. This function manages partial memory deallocation while ensuring proper updates to the super-block's metadata.

9. Deallocate Part of a SuperBlock:
Implement a function to deallocate part of a super-block, freeing a specified number of contiguous blocks within the super-block, starting anywhere within the super-block

void deallocatePartOfSuperBlockAnywhere(int BlockId, int StartIndex, int partSize)

- **Description**: The function deallocates a portion of memory, equal to ***partSize*** from any super-block in a memory pool, based on a given ***BlockId***, starting from a specific ***StartIndex*** within this block. If ***BlockId*** is provided, the function first locates the super-block by traversing the linked list. Using inputs ***StartIndex*** and ***partSize***, it calculates the range for deallocation (***deallocStart*** and ***deallocEnd***) and ensures the deallocation remains within the block's bounds.
  Depending on where the deallocation occurs, the function can either adjust the super-block's size and starting index, fully deallocate the block, or split the block into two smaller blocks if the deallocation occurs in the middle.
  If a new super-block is created from splitting, it is inserted into the linked list, and relevant data is adjusted. If no matching ***BlockId*** is found or the deallocation exceeds the block's boundaries, an error message is displayed.

9. Merge Allocation and Deallocation Functions:
Provide an interface for the user to enter strings, allocate memory, and perform deallocation. Upon each operation, the linked list and memory pool should be updated accordingly.
Function Prototype:

- All allocation and deallocation calls will result in automatic updates to the memory pool and linked list display.

void userMemoryManagementInterface();

- **Description**: This function will serve as the interface where users can input strings, allocate memory, deallocate memory (part or whole), and view the current memory pool status.

Detailed Requirements:

1. **Memory Allocation**:
   - The system should be able to allocate contiguous memory blocks for storing strings of variable lengths.
   - Users should be able to input strings, and appropriate memory will be allocated for them using the allocateSuperBlockForString() function.
2. **Memory Deallocation**:
   - Entire super-blocks or parts of super-blocks should be deallocated as per user input.
   - The linked list and memory pool must be updated accordingly.
3. **Memory Display**:
   - Implement a function to display the current state of the memory pool (DisplayLinkedList()), showing which blocks are free or occupied and displaying the strings stored in the super-blocks.
4. **Edge Cases**:
   - Ensure that no memory overflow or underflow occurs.
   - The system should handle cases where memory allocation fails due to insufficient space.
   - Ensure that partial deallocations are handled correctly, even when they create holes within super-blocks.

Submission Guidelines:

- Submit all source files in C++.
- **Provide a detailed explanation of how each function works in the report.**
- The program must compile and run without errors, and each function should be thoroughly tested with sample inputs.

Bonus: Implement additional features such as:

- Memory defragmentation to merge adjacent empty blocks.
- Dynamic resizing of the memory pool.

Testing and Evaluation:

1. Test the allocation and deallocation functions with various string sizes and memory conditions.
2. Check the handling of edge cases like memory exhaustion and partial deallocation in the middle of a super-block.
3. Validate the program by running multiple allocations and deallocations, observing the expected behavior and output.
4. **You are required to submit output of your code for different allocation/deallocation scenarios.**

   A sample output of a completed code is attached in the end. Ofcourse your TA will check for other use cases, which are not give in the sample output to make sure that your code is running fine.

   Your grade will be calculate as follows:

   a) **50%** marks for a running C/C++ Code and Code Output.
   b) **50%** for viva/quiz, etc.

G:\GIKI\Data Structures and Algorithms\Fall 2024\Codes\Block_Memory_Allocation.exe

```
Memory: ____________________________________________________________________________________________________
****************************************************************************************************
Calling: allocateSuperBlockForString(InputString = S4567890123456789."):
Memory: 1234567890123456789._______________________________________________________________________________
Linked List:
                1: 1234567890123456789.         Indices: ( 0 -> 19),  Size: 20
****************************************************************************************************
Calling: allocateSuperBlockForString(InputString = "ABCDEFGHIJabcdefghi."):
Memory: 1234567890123456789.ABCDEFGHIJabcdefghi._____________________________________________________________
Linked List:
                1: 1234567890123456789.         Indices: ( 0 -> 19),  Size: 20
                2: ABCDEFGHIJabcdefghi.         Indices: ( 20 -> 39),  Size: 20
****************************************************************************************************
Calling: allocateSuperBlockForString(InputString = "9876543210987654321."):
Memory: 1234567890123456789.ABCDEFGHIJabcdefghi.9876543210987654321._________________________________________
Linked List:
                1: 1234567890123456789.         Indices: ( 0 -> 19),  Size: 20
                2: ABCDEFGHIJabcdefghi.         Indices: ( 20 -> 39),  Size: 20
                3: 9876543210987654321.         Indices: ( 40 -> 59),  Size: 20
****************************************************************************************************
Calling: deallocateSuperBlock(BlockId = 1):

Memory deallocated for Super-block id: 1
Memory: ____________________ABCDEFGHIJabcdefghi.9876543210987654321._________________________________________
Linked List:
                2: ABCDEFGHIJabcdefghi.         Indices: ( 20 -> 39),  Size: 20
                3: 9876543210987654321.         Indices: ( 40 -> 59),  Size: 20
****************************************************************************************************
Calling: allocateSuperBlockForString(InputString = "**********."):
Memory: **********._________ABCDEFGHIJabcdefghi.9876543210987654321._________________________________________
Linked List:
                2: ABCDEFGHIJabcdefghi.         Indices: ( 20 -> 39),  Size: 20
                3: 9876543210987654321.         Indices: ( 40 -> 59),  Size: 20
                4: **********.          Indices: ( 0 -> 10),  Size: 11
****************************************************************************************************
Calling: deallocatePartOfSuperBlock(BlockId = 2, partSize = 8)
Part of the super-block deallocated: 8 blocks freed starting from index 20
Memory: **********._________________IJabcdefghi.9876543210987654321._________________________________________
Linked List:
                2: IJabcdefghi.         Indices: ( 28 -> 39),  Size: 12
                3: 9876543210987654321.         Indices: ( 40 -> 59),  Size: 20
                4: **********.          Indices: ( 0 -> 10),  Size: 11
****************************************************************************************************
Calling: eallocatePartOfSuperBlockAnywhere(BlockId = 3, StartIndex = 50, partSize = 5)
Part of the super-block deallocated: 5 blocks freed starting from index 50
Memory: **********._________________IJabcdefghi.9876543210____4321._________________________________________
Linked List:
                2: IJabcdefghi.         Indices: ( 28 -> 39),  Size: 12
                3: 9876543210_____4321.         Indices: ( 40 -> 59),  Size: 20
                5: 4321.                Indices: ( 55 -> 59),  Size: 5
                4: **********.          Indices: ( 0 -> 10),  Size: 11
****************************************************************************************************
Calling: eallocatePartOfSuperBlockAnywhere(BlockId = 2, StartIndex = 30, partSize = 20)
Error: Deallocation exceeds the bounds of the super-block
Memory: **********._________________IJabcdefghi.9876543210____4321._________________________________________
--------------------------------
Process exited after 0.1368 seconds with return value 0
Press any key to continue . . .
```