**Reflection**

The primary purpose of this code is to manage a list of bids using a linked list data structure, providing functionalities for adding, removing, searching for, and displaying bids. Each bid is represented by a `Bid` struct that includes attributes such as `bidId`, `title`, `fund`, and `amount`. The linked list is implemented using dynamic memory allocation, with each node containing a `Bid` and a pointer to the next node. This structure allows for efficient insertions and deletions in constant time when manipulating the head or tail of the list.

The key operations include appending, prepending, removing, and searching for bids. These operations ensure the integrity of the list while dynamically managing memory as nodes are added or removed.

Various techniques were employed to tackle challenges, particularly in memory management and ensuring proper list traversal. The use of dynamic memory allocation for linked list nodes required meticulous attention to avoid memory leaks, necessitating explicit calls to `delete.` One of the most challenging aspects was managing pointers when removing or updating nodes, especially in edge cases such as deleting the head of the list or locating a node in the middle.

Great care was taken to prevent issues such as "dangling pointers" by ensuring that the list is correctly updated after nodes are removed. Furthermore, maintaining consistent naming conventions and carefully handling temporary nodes contributed to improved code readability. By breaking down the logic into smaller functions and employing clear variable names, these challenges were effectively addressed while ensuring code maintainability.

**Pseudocode**

The program manages a list of bids using a linked list.

Each Bid contains details like bidId, title, fund, and amount.

Linked List Operations:

LinkedList Class: Manages the entire list of bids, with methods to add, remove, search, and display bids.

Key Methods in LinkedList:

Append: Adds a bid to the end of the list.

Prepend: Adds a bid to the beginning of the list.

PrintList: Displays all bids in the list.

Remove: Removes a bid from the list by its bidId.

Search: Finds a bid by its bidId and returns it.

Size: Returns the number of bids in the list.

Memory Management:

The program uses dynamic memory allocation to manage the list of bids.

Destructor deletes all nodes to prevent memory leaks when the list is no longer in use.

User Interaction:

The program provides a menu with options:

Enter a new bid.

Load bids from a file.

Display all bids.

Search for a specific bid by ID.

Remove a bid by ID.

Exit the program.

The user can select an option, and the corresponding method will be executed.