**Software Design Report**

**Introduction**

This report shows the design and architecture of a Task Management System developed in C++. The project is structured into various classes, each handling a different aspect of the application, including task creation, task node management, and task list operations with a user interaction menu. The task list has one default task when construct.

**Design Overview**

The software is modular, consisting of several interconnected components:

* Task: Represents an individual task with a content string.
* Node: Acts as a container for tasks, linking them into a doubly-linked list.
* TaskList: Manages a collection of tasks in a list.
* Menu: Provides an interface for user interaction with the task list.

**Task Class**

**Responsibilities:**

* Store the content of a task.
* Provide methods to set and retrieve task content.
* Print task content.

**Attributes:**

* std::string content: The string describing the task.

**Methods:**

* get\_content: Returns the task content.
* set\_content: Sets the task content.
* print\_task: Prints the task content.

**Node Class**

**Responsibilities:**

* Manage the pointers to previous and next nodes.
* Encapsulate a Task object.
* Link and unlink nodes for list manipulation.

**Attributes:**

* Node\* prev: Pointer to the previous node in the list.
* Node\* next: Pointer to the next node in the list.
* Task task: The task contained within the node.

**Methods:**

* Getter and setter for prev and next.
* get\_task: Retrieves the task object.
* link\_prev and link\_next: Methods to link this node with other nodes.
* print\_node: Prints the content of the task within the node.

**Task\_list Class**

**Responsibilities:**

* Maintain the head of the list and a set of all nodes.
* Provide functionality to add, move, and delete tasks within the list.
* Search and check existence of nodes.

**Attributes:**

* Node\* head: Pointer to the first node in the list.
* std::unordered\_set<Node\*> node\_set: A set tracking all nodes for quick existence checking.

**Methods:**

* Getter and setter for head.
* get\_node\_set: Retrieves the set containing all nodes.
* ifNodeExist: Checks if a node exists in the list.
* search\_task: Searches for a task by content.
* add\_front, add\_back: Add tasks to the front/back of the list.
* move\_ahead, move\_back: Move tasks around in the list.
* print\_task\_list: Prints all tasks in the list.
* delete\_list: Deletes all nodes in the list.

**Algorithm for Inserting a New Node into a Doubly Linked Task List:**

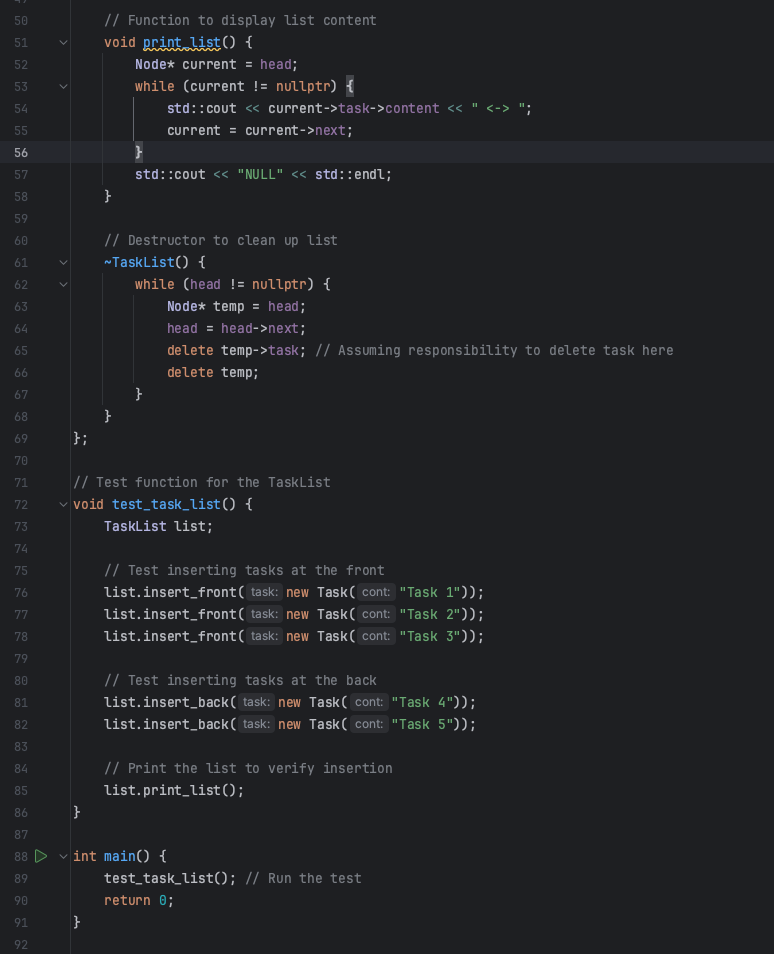
* **Insert at Front:**A screen shot of a computer program

  Description automatically generated
  1. Create a new Node with the Task data.
  2. Set the new\_node->next to the current head of the list.
  3. If the list isn't empty, set head->prev to new\_node.
  4. Update the head to new\_node.
  5. Set new\_node->prev to nullptr.
* **Insert at Back:**A computer screen shot of text

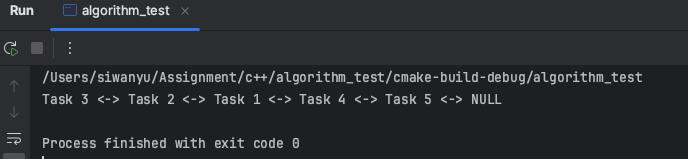
  Description automatically generated
  1. Create a new Node with the Task data.
  2. If the list is empty, make this new\_node the head and tail.
  3. Otherwise, traverse to the end of the list and find the last node.
  4. Set last\_node->next to new\_node and new\_node->prev to last\_node.
  5. Set new\_node->next to nullptr.

**C++ Program to Demonstrate Insertion into a Doubly Linked List:**

**A screen shot of a computer program

Description automatically generated**

**Result:**

****

**Menu Class**

**Responsibilities:**

* Provide a static method to display the menu.
* Run an interactive menu for user commands.
* Has one default task on construct

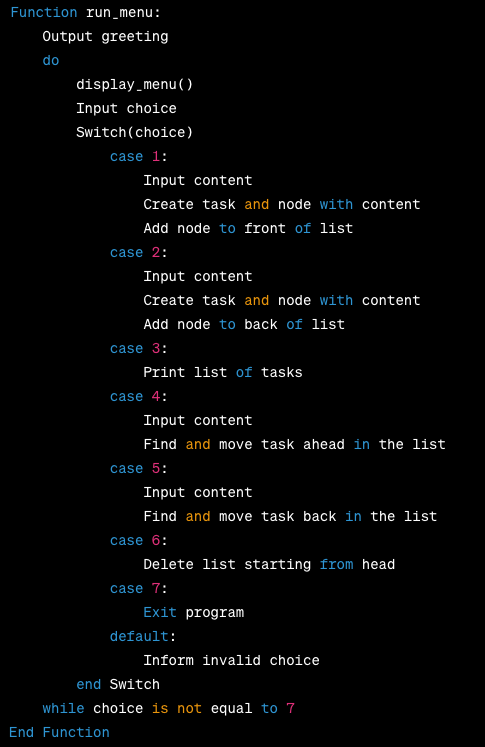
**Attributes:**

* std::string greeting: A welcome message.
* Task essential\_task: A default essential task.
* Node\* essential\_node: A node for the essential task.
* TaskList list: An instance of TaskList to manage tasks.

**Methods:**

* display\_menu: Displays available menu options.
* run\_menu: Handles user input and executes corresponding actions.

Sudo code of run\_menu method :



**Debug**

Syntax error:

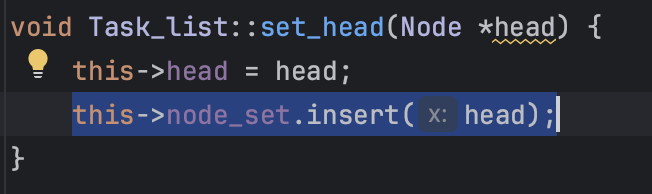
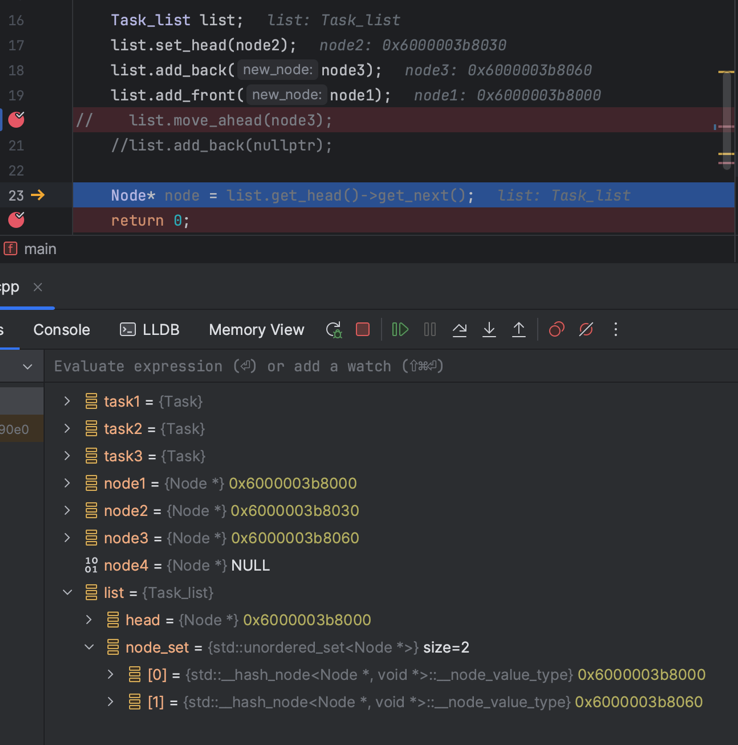
A screen shot of a computer

Description automatically generated

Logial error(Bugs in hash list):

Evaluation:

Added 3 nodes, set node2 as head, then add node3 after node 2, then add node1 before node2. Debug tool showed node1 replaced node2 as head successfully and 3 nodes are in the right sequence as I intended. But in the hash list(store all nodes’ address of task\_list in hash), node2 is not added. That is because the set\_head method need add the code about adding head node into the hash list.



Outcome:A screenshot of a computer code

Description automatically generated

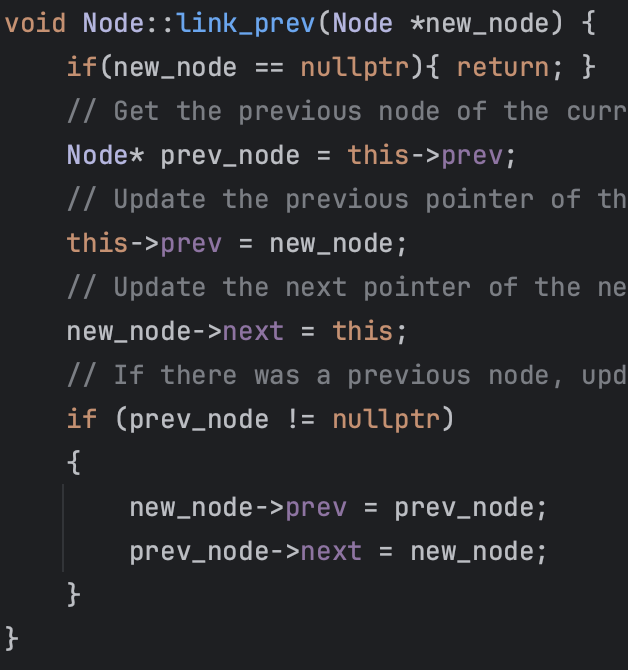
Design error(use node class method link\_prev and link\_next in the definition of task\_list class method move\_front and move\_back):

Evaluation:

After adding 3 nodes, the sequence is node1 node2 node3. Using move\_ahead method to move node3 in front of node2. The ‘prev’ and ‘next’ attribute of node3 has been correctly changed but the ‘next’ attribute of node2 should have been changed to null. This error is caused by using node class method link\_prev and link\_next in the definition of task\_list class method move\_front and move\_back without consider the condition of no node before or after this node(this node is the first or last node of the double-linked list).

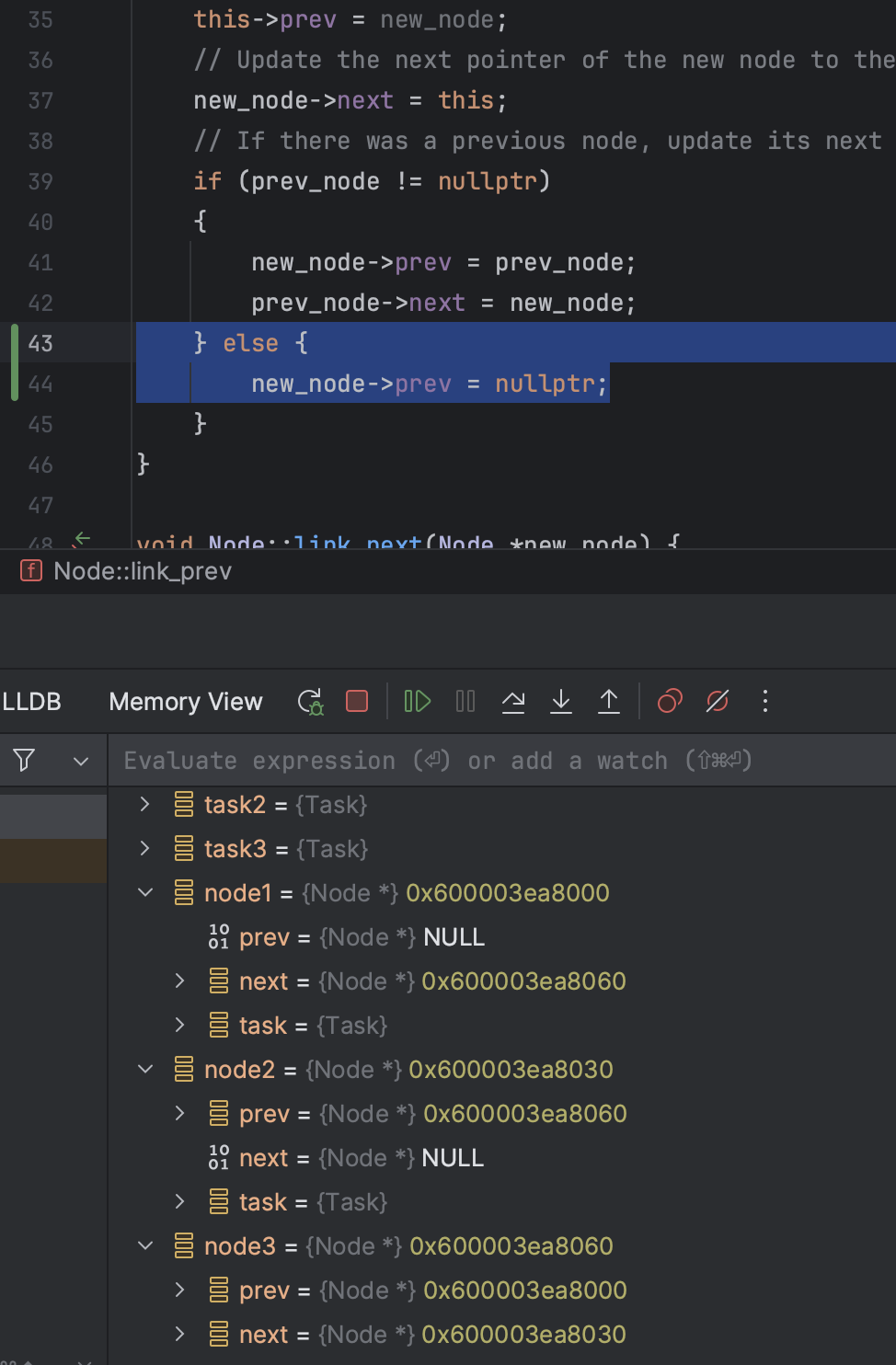
A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

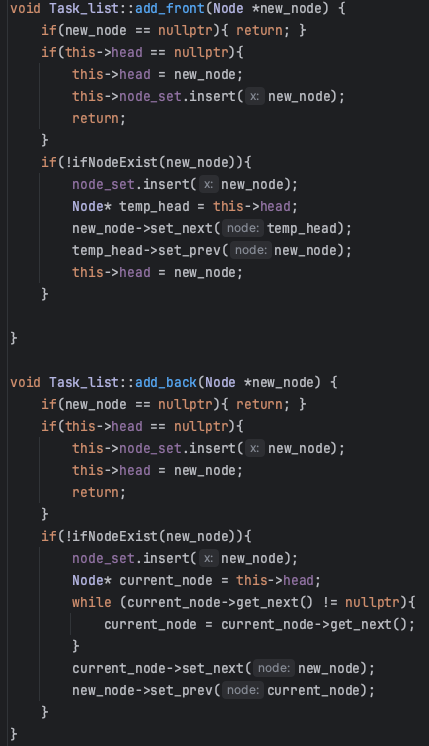
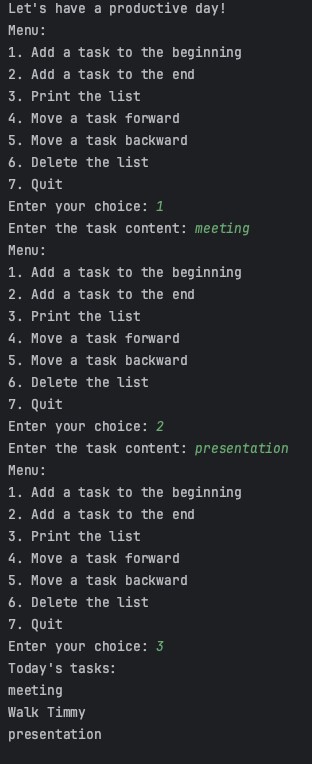
Outcome:

After adding this condition in, the result comes back right.



**Use Case Scenarios**

**Adding Tasks:** Users can add tasks to the beginning or end of the task list using menu options 1 and 2.

Code:test:

**Navigating Tasks:** Users can view tasks and move them forward or backward in the list to prioritize.

Code:A screen shot of a computer code

Description automatically generated

Test:A screenshot of a computer program

Description automatically generated

**Deleting Tasks:** Users have the option to delete the entire list.

Code:A computer screen with text on it

Description automatically generated

Test:A screenshot of a computer program

Description automatically generated

**different techniques to optimise code**

* **Efficient Memory Management:**
  1. The code creates nodes only when needed and properly deletes them in the destructor to avoid memory leaks. This is essential in C++ programming, where manual memory management is crucial for performance and reliability.
* **Avoiding Unnecessary Operations:**
  1. The insert\_back function checks if the list is empty and directly sets the new\_node as head without unnecessary traversal. This is a time-saving optimization when dealing with an empty list.

**C Language Pointer Features Demonstrated in Code**

* **Pointer Variables:** Node class uses pointer variables prev and next to point to the previous and next Node objects in the list, thereby creating a doubly linked list structure.

A black background with white text

Description automatically generated

* **Pointer Operations:** The code performs various pointer operations like assignment, comparison with nullptr, and accessing members of pointed-to structures (arrow operator ->).

A screen shot of a computer code

Description automatically generated

* **Dynamic Memory Management:** The new and delete operations are used to allocate and deallocate memory for Node objects, respectively. This dynamic memory management is crucial in C++ and derived from C's malloc and free.

Node\* essential\_node = new Node(essential\_task); // ... delete node;

**C++ Features**

* **Classes and Encapsulation:** The code defines Task, Node, and Task\_list as classes with private data members and public methods, encapsulating the implementation details and providing a public interface.

A black background with white text

Description automatically generated

* **Templates:** The std::unordered\_set template to store Node\* types, demonstrating generic programming features where the same class can be adapted to store any type.

**Conclusion**

The Task Management System is a console-based application designed to demonstrate object-oriented programming and linked list manipulation in C++.