FACULTY OF COMPUTING
UTM Johor Bahru

SECR1213-05

DATA STRUCTURE & ALGORITHM
Semester 03, 2025/2026

ASSIGNMENT 2
- Dynamic Task Management

Lecturer:

Dr PANG YEE YONG

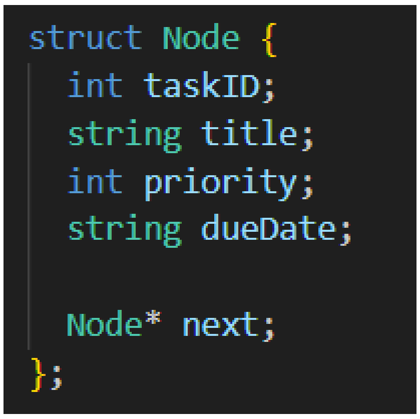
Group Members:

No.	Name	Matric No
1	AHMAD MUNIF BIN BAHARUM	A24CS0038
2	ABDURRAFIQ BIN ZAKARIA	A24CS0031

Table of Contents

PART 1: Implementation

1.1 Node Structure



```
struct Node {
    int taskID;
    string title;
    int priority;
    string dueDate;

    Node* next;
};
```

Figure 1.1 Node Structure

In this code, we construct data sections in one node which data are taskID(int), title(string), priority(int), dueDate(string).

In visual representation:

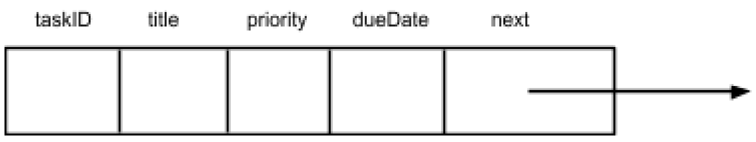
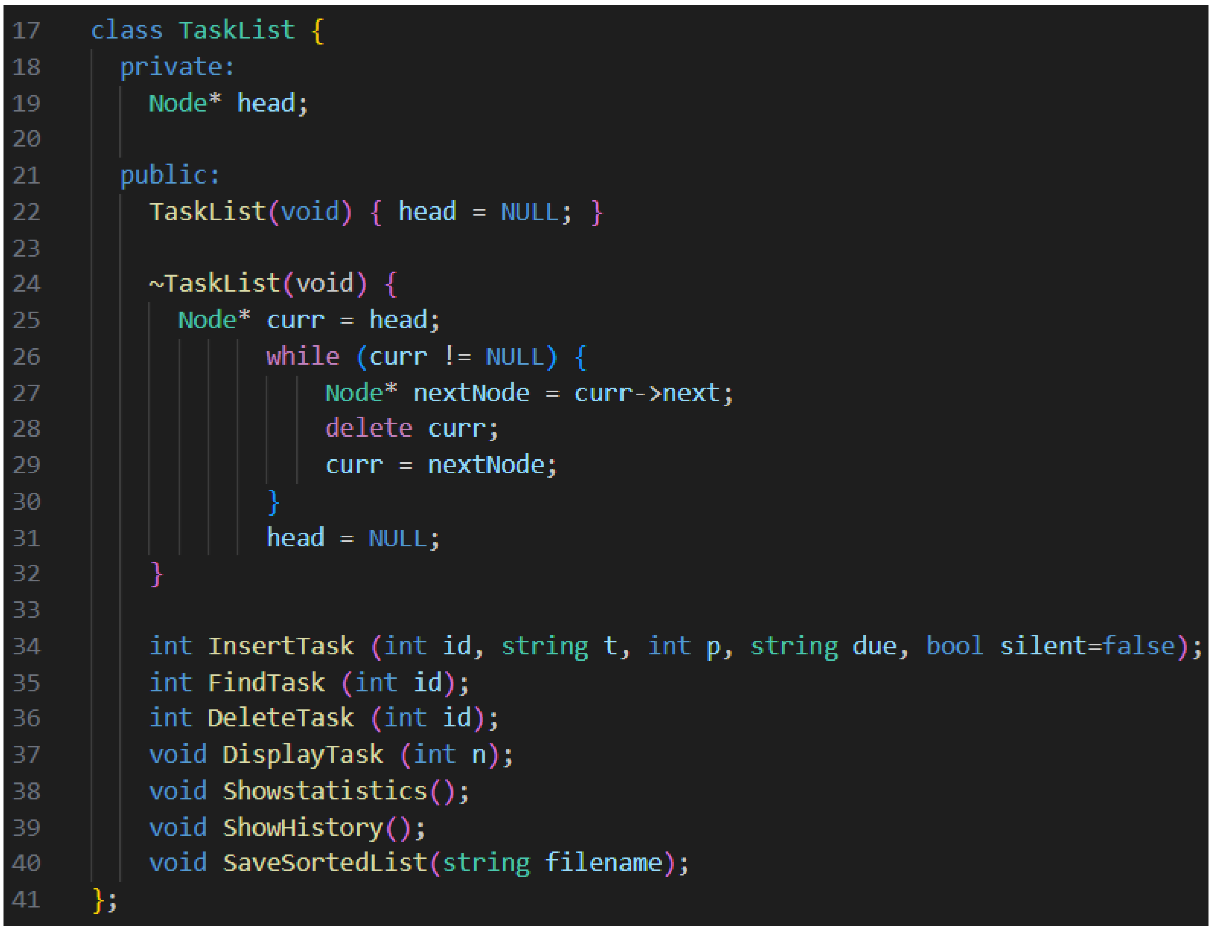


Figure 1.2 Visual representation of node structure

1.2 Class Management



```
17  class TaskList {
18    private:
19      Node* head;
20
21    public:
22      TaskList(void) { head = NULL; }
23
24      ~TaskList(void) {
25        Node* curr = head;
26              while (curr != NULL) {
27                  Node* nextNode = curr->next;
28                  delete curr;
29                  curr = nextNode;
30              }
31              head = NULL;
32      }
33
34      int InsertTask (int id, string t, int p, string due, bool silent=false);
35      int FindTask (int id);
36      int DeleteTask (int id);
37      void DisplayTask (int n);
38      void Showstatistics();
39      void ShowHistory();
40      void SaveSortedList(string filename);
41  };
```

Figure 1.3 Class Management

Encapsulation & Initialization: The TaskList class uses encapsulation by placing the head pointer in the private section. The constructor ensures that when a TaskList object is created, the list starts in a valid empty state by setting head to NULL.

Memory Management: The destructor is implemented to handle manual memory deallocation. It uses a while loop to traverse the linked list, utilizing a temporary nextNode pointer to track the rest of the list while the current node is being deleted. Once all nodes are cleared, head is reset to NULL.

Function Prototypes: Lines 34 through 40 serve as member function prototypes. These declarations notify the compiler of the class's capabilities (like InsertTask or DeleteTask) so they can be called elsewhere in the program before their full implementation is defined.

1.3 Sorting Logic

```
// 1: Insert (sorted by priority, then by taskID)
int TaskList::InsertTask(int id, string t, int p, string due, bool silent) {

 // Check is the new taskID already existed or not
 Node* checkID = head;
 while (checkID != NULL) {
   if (checkID->taskID == id) {
     if (!silent) cout << "Error: Task ID " << id << " already exists!" << endl;
     return 0;
   }
   checkID = checkID->next;
 }

 // if passed taskID check, create & insert all details in newNode
 Node* newNode = new Node;
 newNode->taskID = id;
 newNode->title = t;
 newNode->priority = p;
 newNode->dueDate = due;
 newNode->next = NULL;

 // find the position where to insert the newNode
 // Higher priority comes first
 // BUT if priorities are the same, smaller taskID comes first
 Node* curr = head;
 Node* prev = NULL;
 while (curr != NULL) {
   if (curr->priority > p || (curr->priority == p && curr->taskID < id)) {
     prev = curr;
     curr = curr->next;
   }
   else break;
 }
```

```cpp
 // Adjust the pointers direction(prev & next) to suit the newNode position
 if (prev == NULL) { // Case 1: if the new node becomes the first node
  newNode->next = head;
  head = newNode;
 }
 else { // Case 2: if the new node is in the middle or last
  newNode->next = curr;
  prev->next = newNode;
 }

 RecordHistory("Insert", id);
  if (!silent) cout << "Task " << id << " added to list." << endl;

 return 0;
}

// 2: Search
// Search a task by taskID
// Display the full record if found, otherwise display “Not found”.
// Record the search operation in history (found / not found)
int TaskList::FindTask(int id) {
 Node* curr = head;
 int currIndex = 1;

 while (curr != NULL && curr->taskID != id) { // keep looping until correct ID found
  curr = curr->next;
  currIndex++;
 }

 // display messages for the taskID found or not found
 if (curr) { // display full record if found
  cout << "\n---Task found at position " << currIndex << " ---" << endl;
  cout << "Task ID: " << curr->taskID << endl;
  cout << "Title: " << curr->title << endl;
  cout << "Priority: " << curr->priority << endl;
  cout << "Due Date: " << curr->dueDate << endl;
```

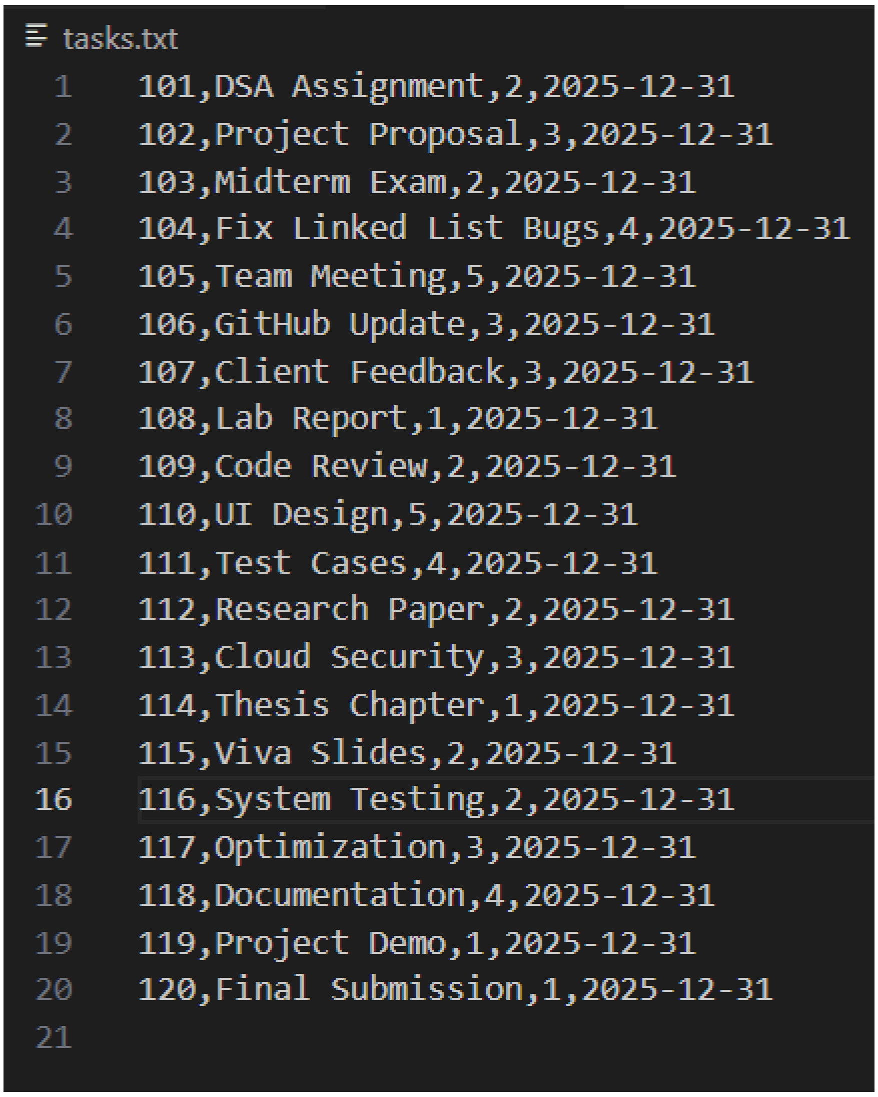
```
    RecordHistory("Search Found", id);
    return currIndex;
  }
  else { // display error if not found
    cout << "\n---Task ID " << id << " not found---" << endl;

    RecordHistory("Search Not Found", id);
    return 0;
  }
}
```

Figure 1.4 Sorting Logic

The InsertTask function maintains a sorted linked list by first verifying that a new taskID is unique to prevent duplication. It then traverses the list using prev and curr pointers to locate the precise insertion point based on a two-tier priority system: tasks with a higher priority value are placed first, and in cases of equal priority, the task with the smaller taskID takes precedence. Finally, the function adjusts the node pointers to integrate the newNode into the sequence and logs the event to the history file.

PART 2: Sample File Content

2.1 Sample Content of tasks.txt



tasks.txt

```
101,DSA Assignment,2,2025-12-31
102,Project Proposal,3,2025-12-31
103,Midterm Exam,2,2025-12-31
104,Fix Linked List Bugs,4,2025-12-31
105,Team Meeting,5,2025-12-31
106,GitHub Update,3,2025-12-31
107,Client Feedback,3,2025-12-31
108,Lab Report,1,2025-12-31
109,Code Review,2,2025-12-31
110,UI Design,5,2025-12-31
111,Test Cases,4,2025-12-31
112,Research Paper,2,2025-12-31
113,Cloud Security,3,2025-12-31
114,Thesis Chapter,1,2025-12-31
115,Viva Slides,2,2025-12-31
116,System Testing,2,2025-12-31
117,Optimization,3,2025-12-31
118,Documentation,4,2025-12-31
119,Project Demo,1,2025-12-31
120,Final Submission,1,2025-12-31
```

Figure 2.1 Content of tasks.txt

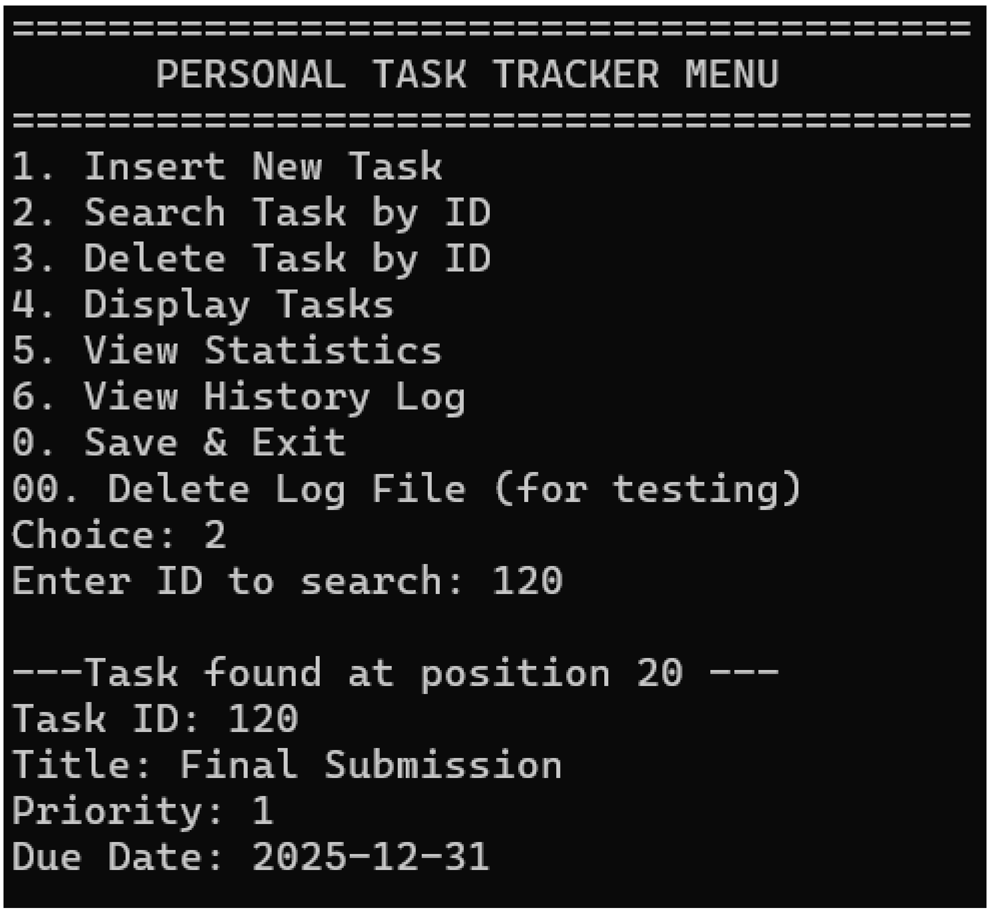
This sample content is from the code that has been given in the question for sample task code generation. After the code has been generated, It saves the data in a file named tasks with extension ".txt". This file is stored in the same folder the code placed.

The content of this file is 20 pieces of given data and yet still not sorted in order of priority but only sorted by increasing number of taskID.

PART 3: Functional Testing & Output

3.1 Task Search

3.1.1 Search (Found)



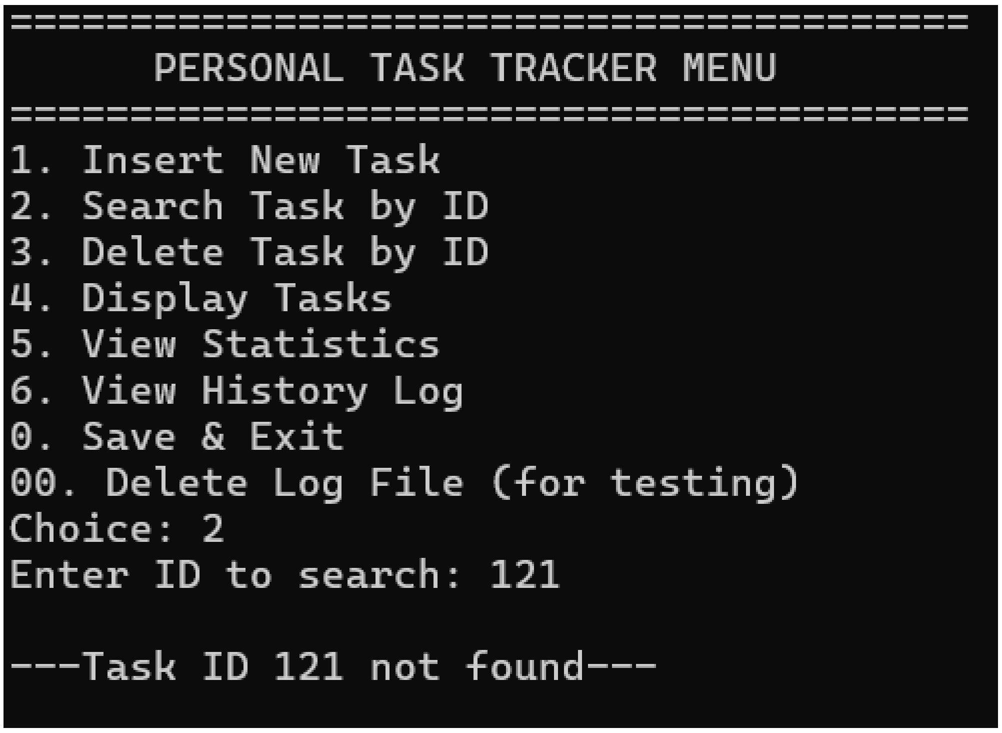
```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 2
Enter ID to search: 120

---Task found at position 20 ---
Task ID: 120
Title: Final Submission
Priority: 1
Due Date: 2025-12-31
```

Figure 3.1 Search (Found)

This display output shows that when we choose number 2 in the menus given, that means we chose the option Search Task by ID. Then, the program asked us to put the ID to search and we chose ID 120.

The program says that they found taskID 120 at the position 20 in the file tasks_sorted.txt that has been sorted by increasing number of priority. It also shows the data in what it has for taskID 120.

3.1.2 Search (Not Found)



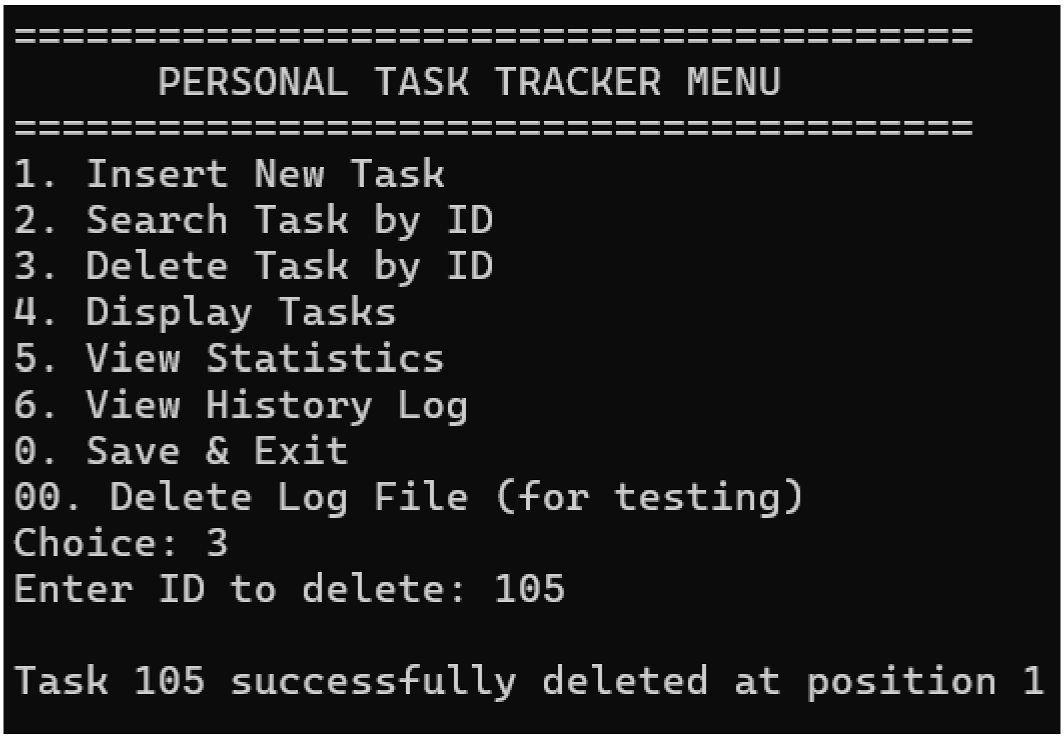
```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 2
Enter ID to search: 121

---Task ID 121 not found---
```

Figure 3.2 Search (Not Found)

This display output shows that when we choose number 2 in the menus given, that means we chose the option Search Task by ID. Then, the program asked us to put the ID to search and we chose ID 121 on purpose. The program says that it is not found in the data file for taskID 121. Then, the program shows that it is not found by displaying "---Task ID 121 not found ---".

3.2 Task Deletion

3.2.1 Delete (Head)

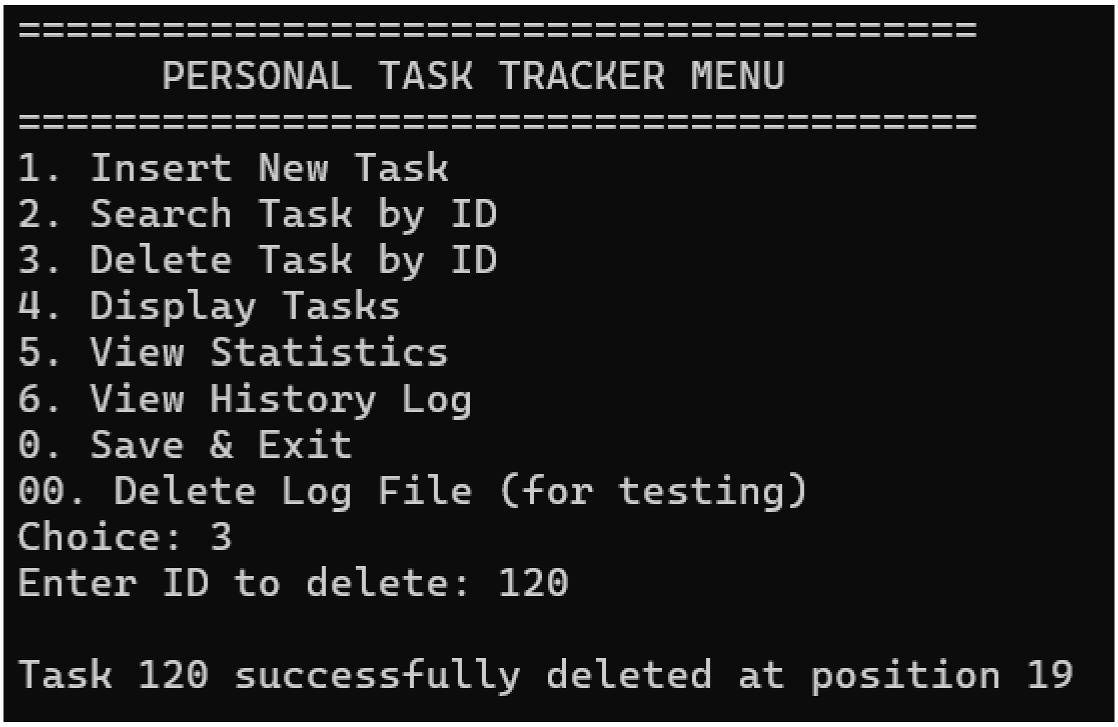


```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 3
Enter ID to delete: 105

Task 105 successfully deleted at position 1
```

Figure 3.3 Delete (Head)

When we choose 3 from the menus given, the program will execute for option Delete Task by ID. Then, the program asks the user to enter which taskID user wants to delete. We chose taskID 105 since that ID is the head or the position first in the sorted file data. After that, it shows it successfully deleted by displaying "Task 105 successfully deleted at position 1".

3.2.2 Delete (Tail)



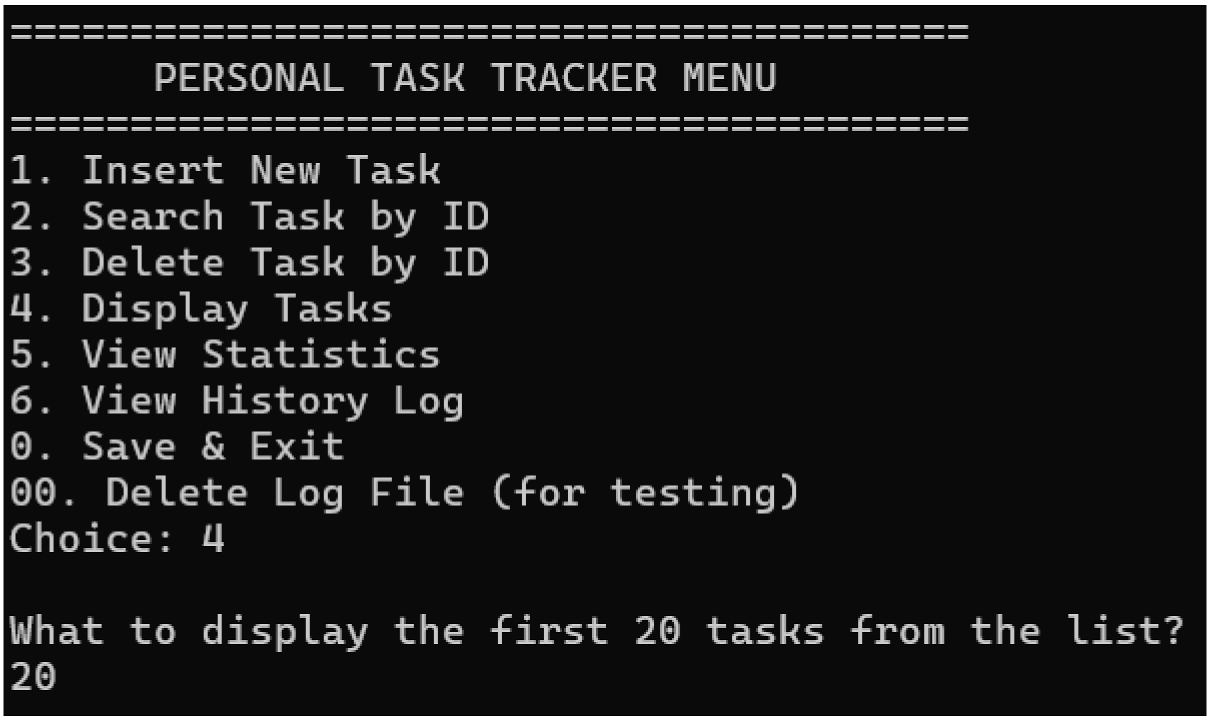
```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 3
Enter ID to delete: 120

Task 120 successfully deleted at position 19
```

Figure 3.4 Delete (Tail)

When we choose 3 from the menus given, the program will execute for option Delete Task by ID. Then, the program asks the user to enter which taskID user wants to delete. We chose taskID 120 since that ID is the tail or the position last in the sorted file data. After that, it shows it successfully deleted by displaying “Task 120 successfully deleted at position 19”. It shows position 19 because currently the data in file now is 19 because we already deleted the taskID 105.

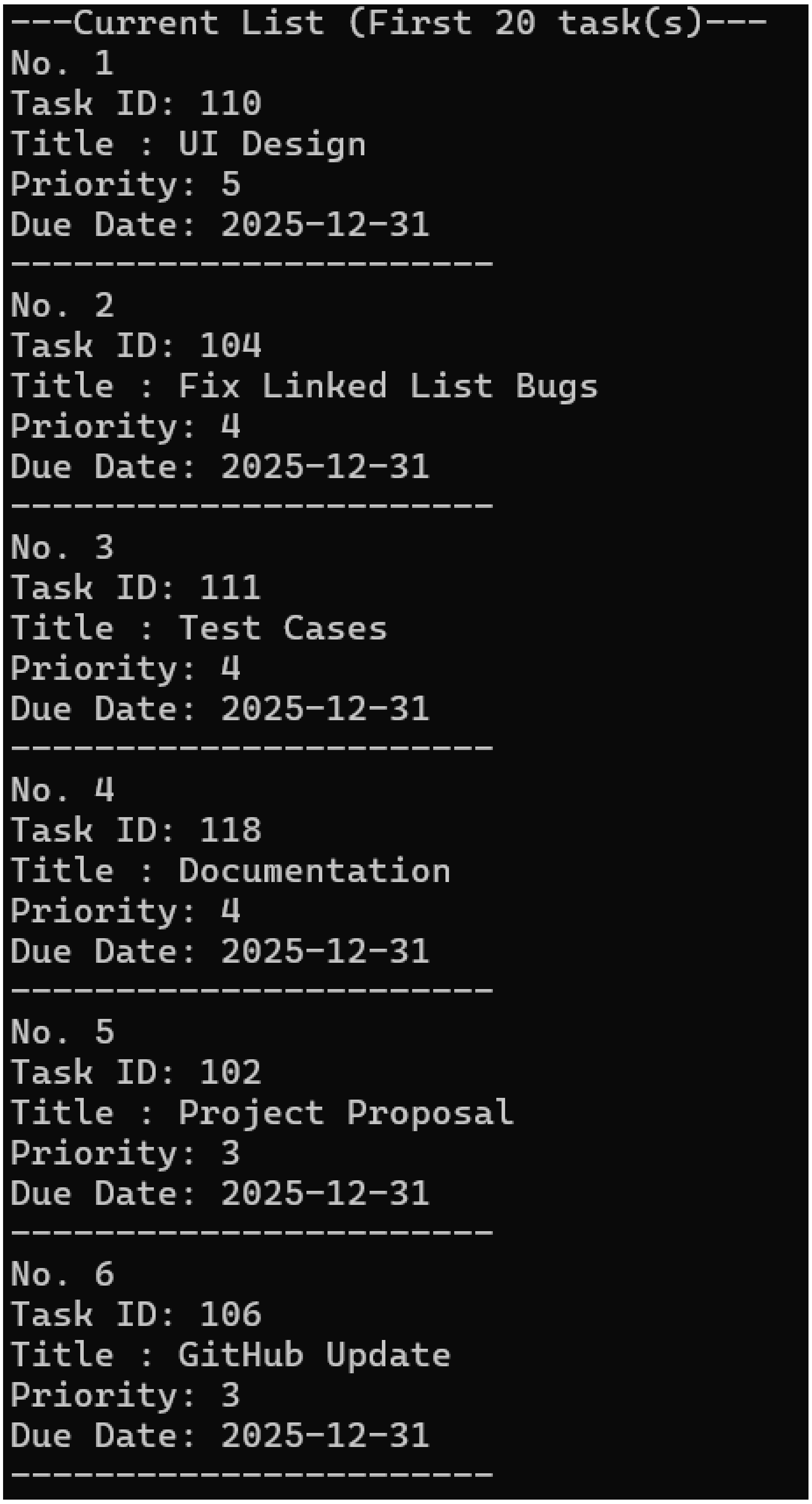
3.3 Display & Statistics

3.3.1 Display (After Deletion) & Total Number of Tasks After Deletion



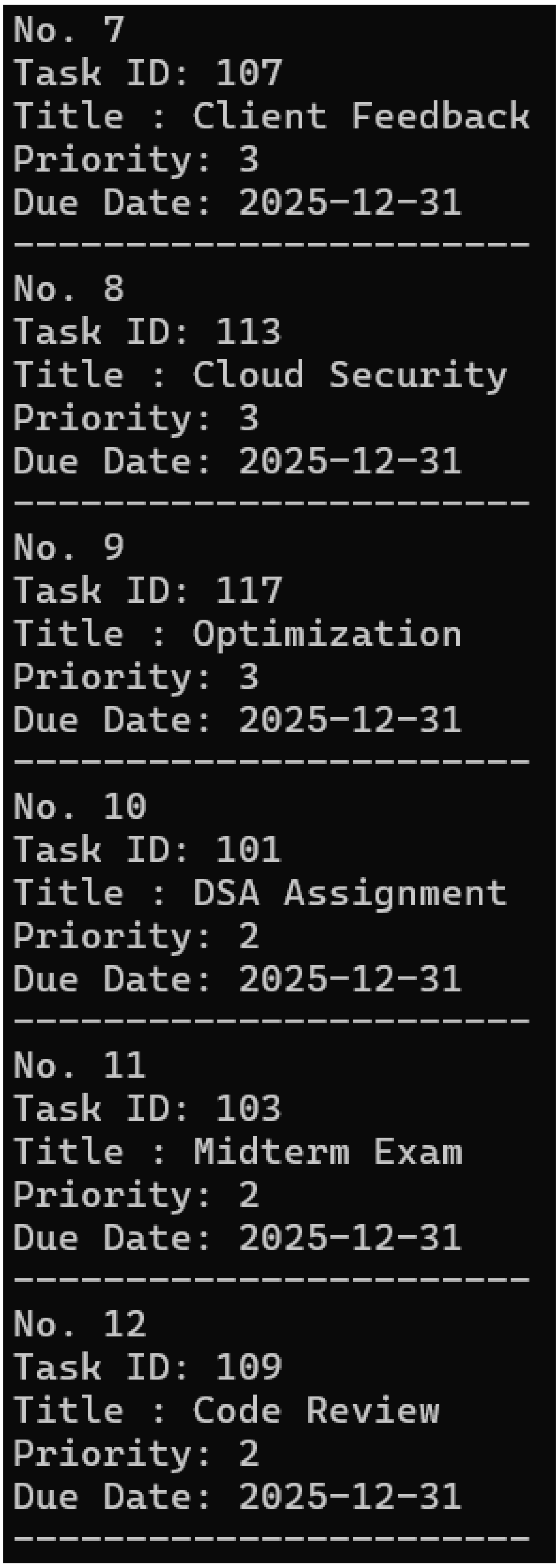
```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 4

What to display the first 20 tasks from the list?
20
```

Figure 3.5.1 Display (After Deletion)



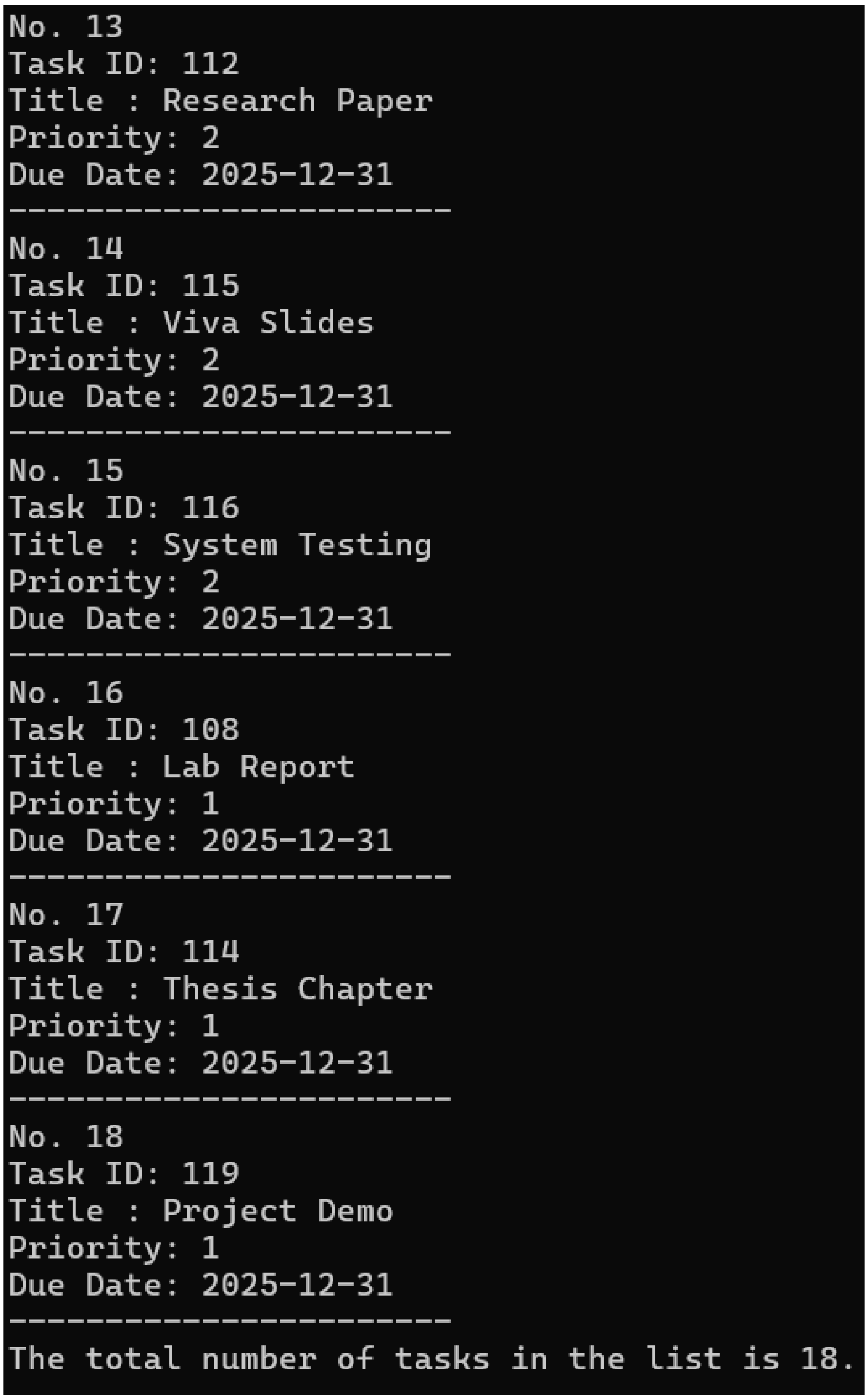
```
---Current List (First 20 task(s)---
No. 1
Task ID: 110
Title : UI Design
Priority: 5
Due Date: 2025-12-31
-----------------------
No. 2
Task ID: 104
Title : Fix Linked List Bugs
Priority: 4
Due Date: 2025-12-31
-----------------------
No. 3
Task ID: 111
Title : Test Cases
Priority: 4
Due Date: 2025-12-31
-----------------------
No. 4
Task ID: 118
Title : Documentation
Priority: 4
Due Date: 2025-12-31
-----------------------
No. 5
Task ID: 102
Title : Project Proposal
Priority: 3
Due Date: 2025-12-31
-----------------------
No. 6
Task ID: 106
Title : GitHub Update
Priority: 3
Due Date: 2025-12-31
-----------------------
```

Figure 3.5.2 Display (After Deletion)



```
No. 7
Task ID: 107
Title : Client Feedback
Priority: 3
Due Date: 2025-12-31
-----------------------
No. 8
Task ID: 113
Title : Cloud Security
Priority: 3
Due Date: 2025-12-31
-----------------------
No. 9
Task ID: 117
Title : Optimization
Priority: 3
Due Date: 2025-12-31
-----------------------
No. 10
Task ID: 101
Title : DSA Assignment
Priority: 2
Due Date: 2025-12-31
-----------------------
No. 11
Task ID: 103
Title : Midterm Exam
Priority: 2
Due Date: 2025-12-31
-----------------------
No. 12
Task ID: 109
Title : Code Review
Priority: 2
Due Date: 2025-12-31
-----------------------
```

Figure 3.5.3 Display (After Deletion)



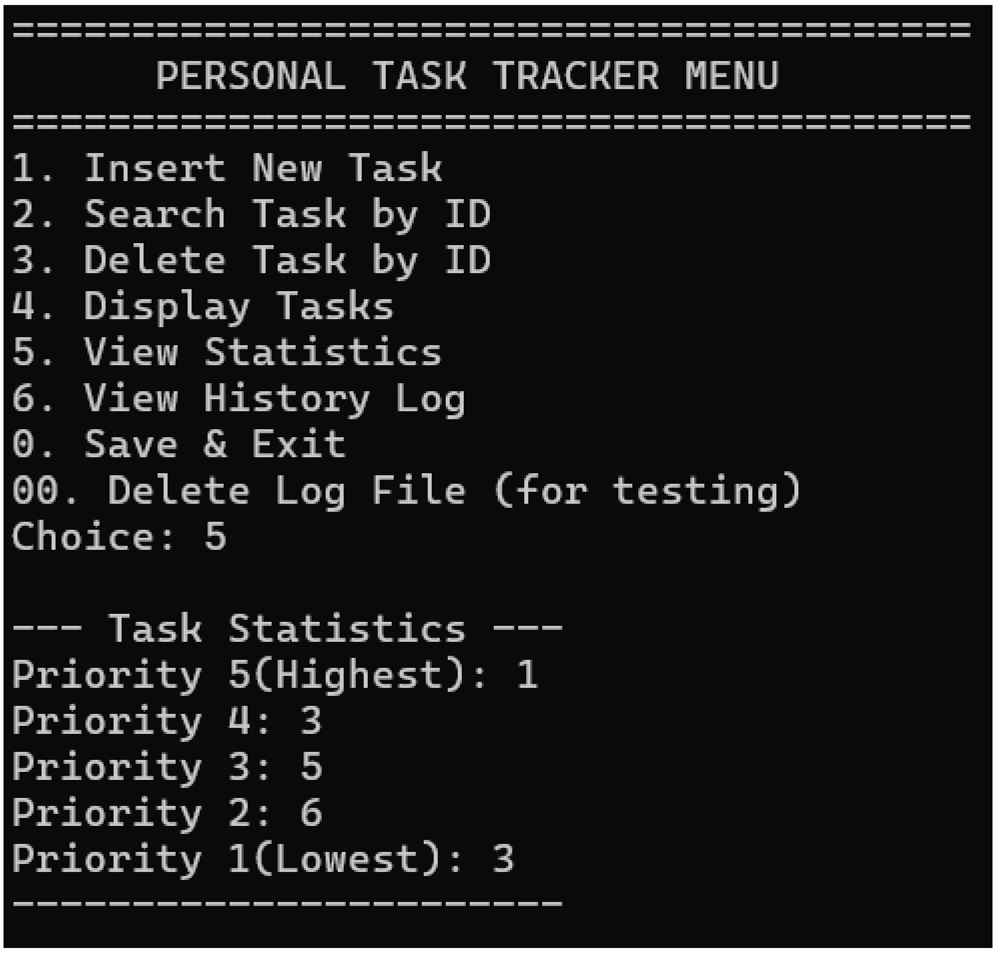
```
No. 13
Task ID: 112
Title : Research Paper
Priority: 2
Due Date: 2025-12-31
-----------------------
No. 14
Task ID: 115
Title : Viva Slides
Priority: 2
Due Date: 2025-12-31
-----------------------
No. 15
Task ID: 116
Title : System Testing
Priority: 2
Due Date: 2025-12-31
-----------------------
No. 16
Task ID: 108
Title : Lab Report
Priority: 1
Due Date: 2025-12-31
-----------------------
No. 17
Task ID: 114
Title : Thesis Chapter
Priority: 1
Due Date: 2025-12-31
-----------------------
No. 18
Task ID: 119
Title : Project Demo
Priority: 1
Due Date: 2025-12-31
-----------------------
The total number of tasks in the list is 18.
```

Figure 3.5.4 Display (After Deletion)

When we choose 4 from the menus given, the program will execute the option Display Task. The programs ask the user how many tasks they want to be listed firstly? So I chose 20 lists.

So, the program shows the list for 18 lists because we already deleted 2 taskID before this. The program also counts the list and shows the total number of tasks in the list is 18.

3.3.2 Statistic Of Priorities



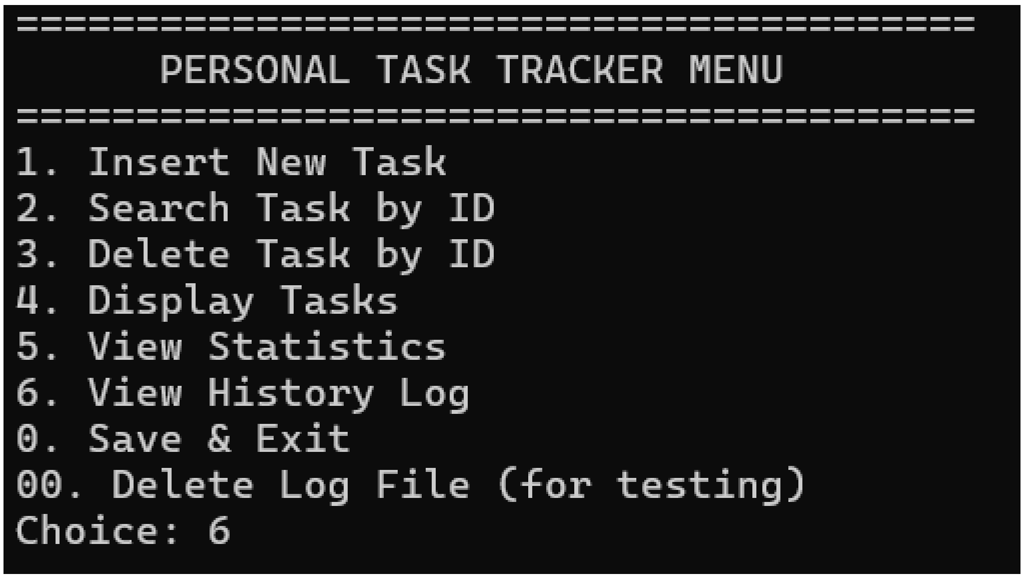
```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 5

--- Task Statistics ---
Priority 5(Highest): 1
Priority 4: 3
Priority 3: 5
Priority 2: 6
Priority 1(Lowest): 3
-----------------------
```

Figure 3.6 Statistic of Priorities

I choose number 5 from the menus given for the option View Statistics. Then, the program starts showing the statistics of priority, which is priority 5 is the highest priority that should be taken care of (has 1 task) and priority 1 is the lowest priority that can be ignored for a while (has 3 tasks).

PART 4: Audit & History Logs

4.1 History Logs Display

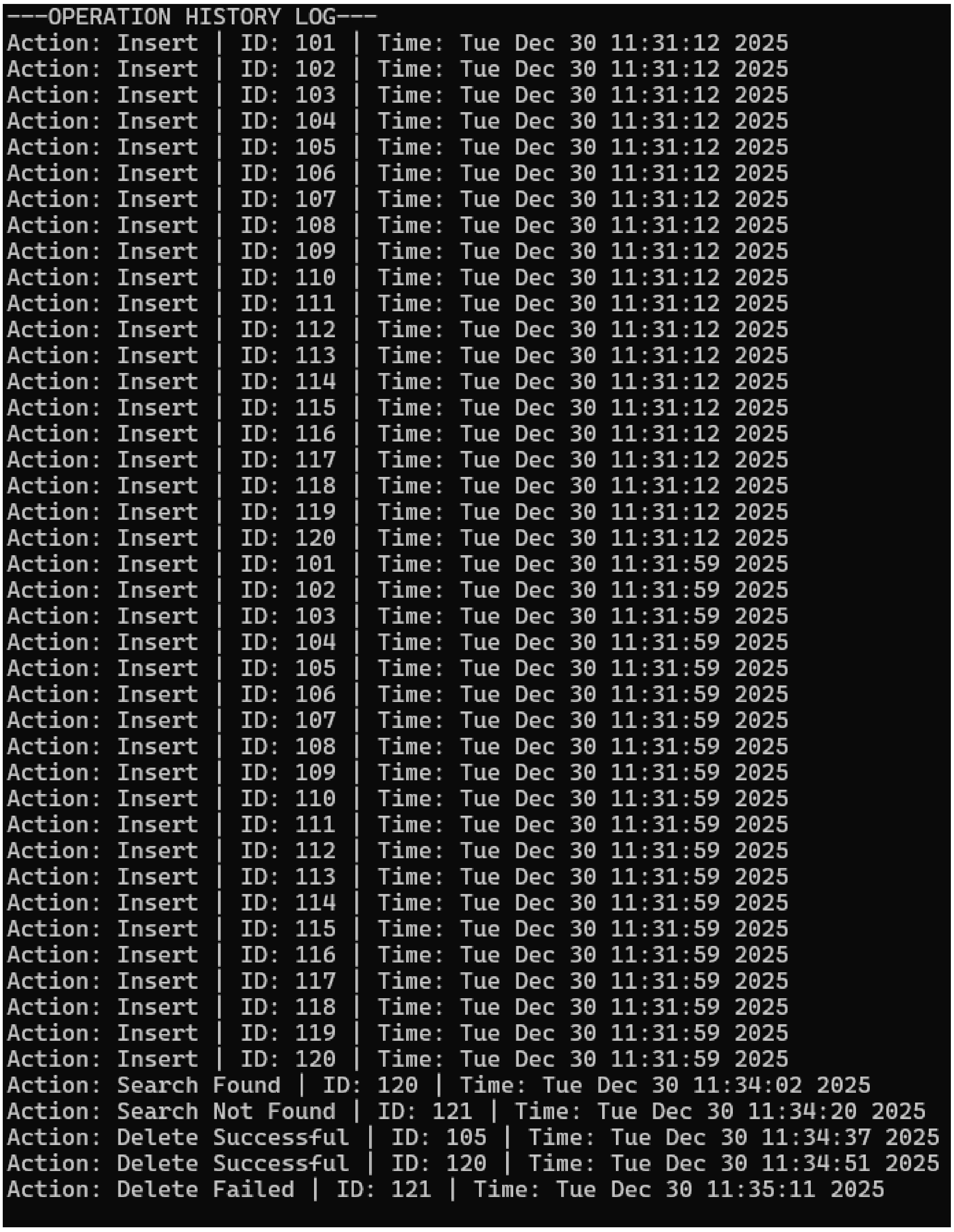


```
========================================
      PERSONAL TASK TRACKER MENU
========================================
1. Insert New Task
2. Search Task by ID
3. Delete Task by ID
4. Display Tasks
5. View Statistics
6. View History Log
0. Save & Exit
00. Delete Log File (for testing)
Choice: 6
```

Figure 4.1.1 History Logs Display

For this time, we chose option number 6 for option View History Log because we want to see what we are doing all this time in this program. Basically, we want to observe the “action”, “ID” and “timestamp”. By doing this, we can ensure that we don't cheat while doing our task.

For the option “00. Delete Log File (for testing)”, we put that option on the menu because we want to reset the log to be clear before we run the next program. This can ensure the good log to be present in this report. It just functions for us while doing the testing session to find any error.



```
---OPERATION HISTORY LOG---
Action: Insert | ID: 101 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 102 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 103 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 104 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 105 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 106 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 107 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 108 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 109 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 110 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 111 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 112 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 113 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 114 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 115 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 116 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 117 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 118 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 119 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 120 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 101 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 102 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 103 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 104 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 105 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 106 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 107 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 108 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 109 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 110 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 111 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 112 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 113 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 114 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 115 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 116 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 117 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 118 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 119 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 120 | Time: Tue Dec 30 11:31:59 2025
Action: Search Found | ID: 120 | Time: Tue Dec 30 11:34:02 2025
Action: Search Not Found | ID: 121 | Time: Tue Dec 30 11:34:20 2025
Action: Delete Successful | ID: 105 | Time: Tue Dec 30 11:34:37 2025
Action: Delete Successful | ID: 120 | Time: Tue Dec 30 11:34:51 2025
Action: Delete Failed | ID: 121 | Time: Tue Dec 30 11:35:11 2025
```

Figure 4.1.2 History Logs Display

4.2 Verification of tasks.txt

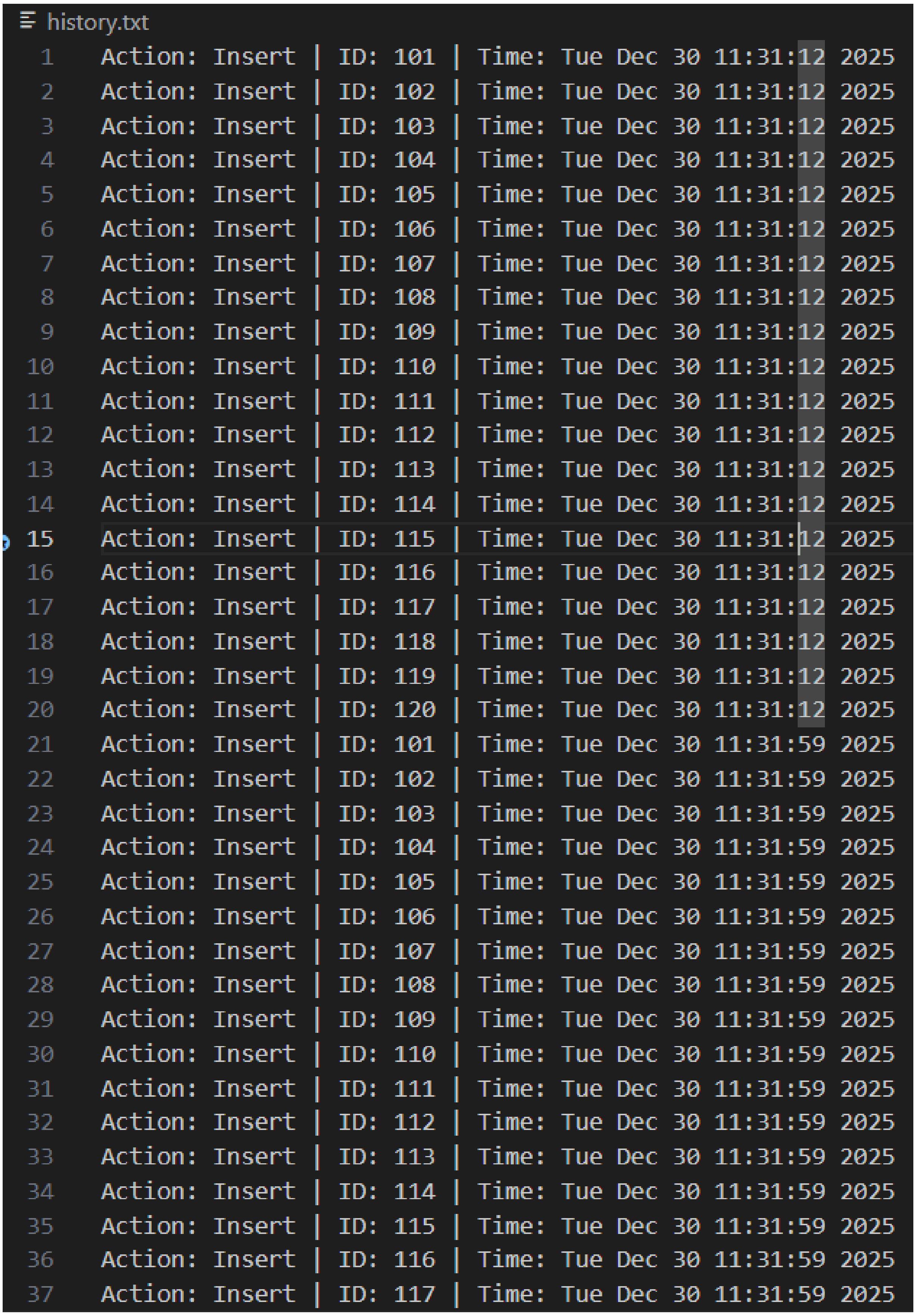
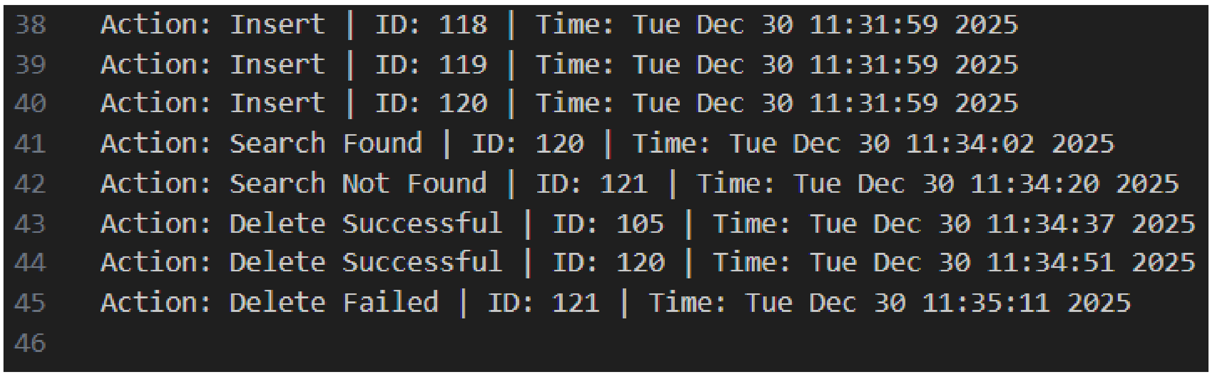

history.txt
```
Action: Insert | ID: 101 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 102 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 103 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 104 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 105 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 106 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 107 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 108 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 109 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 110 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 111 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 112 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 113 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 114 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 115 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 116 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 117 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 118 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 119 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 120 | Time: Tue Dec 30 11:31:12 2025
Action: Insert | ID: 101 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 102 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 103 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 104 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 105 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 106 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 107 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 108 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 109 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 110 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 111 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 112 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 113 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 114 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 115 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 116 | Time: Tue Dec 30 11:31:59 2025
Action: Insert | ID: 117 | Time: Tue Dec 30 11:31:59 2025
```

Figure 4.1.3 History Logs Display



```
38   Action: Insert | ID: 118 | Time: Tue Dec 30 11:31:59 2025
39   Action: Insert | ID: 119 | Time: Tue Dec 30 11:31:59 2025
40   Action: Insert | ID: 120 | Time: Tue Dec 30 11:31:59 2025
41   Action: Search Found | ID: 120 | Time: Tue Dec 30 11:34:02 2025
42   Action: Search Not Found | ID: 121 | Time: Tue Dec 30 11:34:20 2025
43   Action: Delete Successful | ID: 105 | Time: Tue Dec 30 11:34:37 2025
44   Action: Delete Successful | ID: 120 | Time: Tue Dec 30 11:34:51 2025
45   Action: Delete Failed | ID: 121 | Time: Tue Dec 30 11:35:11 2025
46
```

Figure 4.1.4 History Logs Display

As you can see from the log history, let's focus from line 21 until line 45. For line 21 to line 40, the program records the data inserted for the data generation that has been given from the question around 20 data. At line 41, we did a search activity and we successfully found what we searched for. For line 42, It shows that it has not found what we are looking for, which is ID 121 that does not exist in the file.

For the line 43 until line 44, we did the deletion activity to test if we can delete the data from the top and from the bottom using the node in singly linked list. Line 45 shows that the delete failed because we want to try if we can delete the non-existent taskID.