**Console-Based Email (using C++)**

**Group Number:** 18

**Name of the Group Members**

Chikkeri Chinmaya (Roll No. 211IT017)

Harikiran H (Roll No. 211IT023)

Umesh (Roll No. 211IT073)

Vishwa Mohan Reddy G (Roll No. 211IT082)

Vismay P (Roll No. 211IT083)

|  |
| --- |
| **What is the Problem?** |
| Messages and emails have become important for communication because they allow users to send information in the click of a button but there are still quite a few problems which are associated with the current email systems some of them are listed below   * **Delayed or lost Emails:**   Delayed or lost emails are not only annoying, but they can also get you in trouble. Losing precious client-related information is one of the best ways to ruin business reputation instantly   * **Emails stuck in outbox**   Emails stuck in the outbox tab are another common yet annoying bug. This is different from the situation described above, where the email delivery is delayed, but the message isn't in the outbox folder.   * **Bad Hosting and disk quota errors**   One often face slow email delivery and full inboxes, this is due to more unwanted advertisements coming in and unwanted emails that will be clustered in inbox.   * **Misusing Messages and Data Privacy**   Here, we are not talking about technical issues with mails but using them in a way that negatively affects productivity from the hosts. Unfortunately, this problem is faced by most users where their privacy was compromised and no action has been taken. |
|  |
|  |
| **What is the Possible Solution?** |
| The possible solutions to the problems discussed above:   * Swift Email system without much delay can be implemented which enhances text based messaging experience for users and helps in faster communication * The star and the inbox numbering help the user to keep their inbox organized and helps in the faster accessibility of required mails and to prioritize them. * The delete for me feature helps them to get rid of unwanted messages * A simple, easy-to-use email system which is straightforward and user-friendly is needed to be developed. * The privacy of the user is should not be compromised and there must be no third party interference in the middle to complete the mail transfer and authentication should be needed to login into account   In this mini project we have tried to resolve the issues by building a “Console based Email using C++”. |
|  |
| **Which Data Structure to Use for the Solution (Details of the Data Structure)** |
| The Data Structure to be used for the solution is the linked list:  A linked list is a type of data structure in which each element, called a node, stores a value and a reference to the next and the previous element in the list. Linked lists are useful in certain situations, because they can allow for efficient insertion and deletion of elements anywhere in the list.  There are two types of linked lists: singly and doubly linked lists.  A singly linked list only contains a reference to the next node in the list.  A doubly linked list, on the other hand, contains references to both the next and previous elements, allowing for traversal in both directions.  Data Structures In The Real World — Linked List | by Christopher Webb |  Journey Of One Thousand Apps | Medium |
|  |
|  |
| **Justify Why This is the Best Data Structure for this Application?**  *(Minimum 150 words or more, better to list the justifications point-wise, include figure and tables if needed)* |
| Linked lists is the best data structure to use because of the following reasons   * Dynamic data storage: Linked lists do not have a fixed size and can grow or shrink as needed. This is useful for a email application, where the number of emails are being sent and received over time. * Efficient Insertions and Deletions: In a linked list, it's easy to insert or delete elements anywhere in the list which is crucial for a email application as it should be able to send and delete mails efficiently. * Email Threads: A linked list can be used to store mails in a thread format where each node represents a message and link to the next message. It's useful for the users to see the continuity in the inbox. * Memory efficiency: Linked list are space efficient since they store a reference to the next element and the previous elements. It will be a good fit for a console-based messenger as it need not to store any graphical information and can save memory. * Simple to implement: Linked lists are simple to implement and understand, which makes it an ideal choice for a console-based E-mail, where the complexity should be kept minimal. |
|  |
|  |
| **Implementation Details (Experimental Setup)** |
| Implementing a console-based email involved creating several classes to handle different aspects of the application, such as users and mails. A class for the user stored information such as the user's name and password, as well as pointers to the head of singly linked lists for sent and received mails  A class for messages stores information such as the sender, recipient, message text, and timestamps. A pointer to the next message in the linked list would also be stored, allowing for efficient traversal through the list of messages. The messages class also is able to store flags for read, star and delete status.  To send a message, the user inputs the recipient's username, message text, and then the message gets added to the recipient's received messages linked list and the sender's sent messages linked list. To read a message, the user must navigate to their received messages list, and then select the message they wish to read. The message's read flag would be set to true once the user has read it.  To delete a message, the user should navigate to their sent or received messages list, and then select the message they wish to delete. The message would be removed from the linked list and stored in a trash vector for recovery or permanent deletion.  For searching messages, a linear search through the linked list of sent and received messages is be implemented, searching for a specific sender or recipient.  To manage trash, users have options like reading the message, permanently deleting the message or restoring the message to the original linked list.  The overall implementation involved creating classes to handle user and message data, implementing functions to handle sending, receiving, reading, and deleting messages, and providing a user-friendly interface for navigating and interacting with the application through command-line input and output. The console-based interface would be the main point of interaction for the users, displaying menus and options for the user to navigate through the application. |
|  |
|  |
| **Complexity/Performance Analysis** |
| To get necessary actions we have mainly used the concept of linked list – traversal, deletion, insertion and searching. The time complexity of each concept is discussed below:  **Searching:** When searching for an element in a singly linked list, the search must begin at the head of the list and iterate through each element in the list, one by one, until the element is found. If the element is near the end of the list, the search could take O(n) time [Worst case time complexity], where n is the number of elements in the list.  **Insertion:** Inserting an element at the head of a singly linked list can be done in constant time O(1) [Best case Time Complexity] since we only have to update the head and tail pointers of the list. Inserting an element at the end of the list is also O(1) if the tail pointer is maintained. However, inserting an element at a specific position in the middle of the list requires iterating through the list to find the position and then insert, which takes O(n) time.  **Deletion:** Deleting an element from a singly linked list is similar to searching for an element, where the list must be iterated through to find the element. Once the element is found, its previous node's next pointer is updated to point to the node following the element to be deleted. This process takes O(n) time. |
|  |
|  |
| **Observations and Conclusions** |
| The observations and conclusion are   * The program was successful in implementing the core functionality of sending and receiving emails. * The user interface was effective in allowing the user to navigate and manage their inbox. * Any additional features that were implemented, such as email searching or deleting, were well-received by users and added value to the program. * The email messenger was able to fulfil its intended purpose of providing a convenient way for users to access and read the mail, star the important mail and keep track of sent messages from the console. * Overall, the console-based email messenger was a useful and functional tool for managing mails. |