**MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING**

*Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution*

*(A Unit of Rajalaxmi Education Trust®, Mangalore - 575001)*

*Affiliated to V.T.U., Belagavi, Approved by AICTE, New Delhi*



**TECHNICAL TRAINING PROJECT**

**Group-CP055**

**TOPIC : EMAIL CLIENT**

**GROUP MEMBERS:**

|  |  |
| --- | --- |
| Srajan N Shetty | 4MT21AI056 |
| B Mahesh Kumar Shetty | 4MT21AI012 |
| Abhinav S Kangale | 4MT21AI002 |
| Abhin Shetty | 4MT21AI001 |
| Prajwal Rai | 4MT21AI037 |

**TABLE OF CONTENTS**

**Problem Statement**

**1.Abstract**

**2.Introduction**

**2.1 Background**

**2.2 Objectives**

**3.Technologies Used: C**

**4.System Architecture**

**4.1 Front-End**

**4.2 Back-End**

**4.3 Database**

**5.Project Modules**

**5.1 Module 1: Email Client**

**5.2 Module 2: Menu Handling**

**5.3 Module 3: File Manipulation**

**6.Features and Functionality**

**6.1 Feature 1: Compose Email**

**6.2 Feature 2: View Email**

**6.3 Feature 3: Mark Email for Deletion**

**6.4 Feature 4: Delete Marked Emails**

**8.Testing**

**8.1 To check If the email is present or not**

**8.2 To view the email in the directory**

**8.3 To delete the marked email**

**10.Future Enhancements**

**10.1 Time at which email is created/deleted**

**10.2 Integrated Task Management**

**11.Conclusion**

**12.References**

**13.Appendices**

**13.1 Screenshots**

# 13.2 Code Snippets

**Problem Statement :**

In this project, you are tasked with designing and implementing a basic email client in the C programming language. The email client should be capable of sending, receiving, and managing emails, and it should provide the following functionalities:

## 2.2 Objectives

Composing Emails: Users should be able to compose new emails. The email composition should include fields for the recipient's email address, subject, message body, and the option to attach files (attachments can be text files, images, or other common formats).

Replying to Emails: Users should be able to reply to received emails. The reply should include the original message as a quote, and users can compose their responses.

Forwarding Emails: Users should be able to forward received emails to other recipients. The forwarded email should include the original message and allow users to add additional comments if needed.

Receiving Emails: The email client should be able to retrieve emails from a remote email server using standard email protocols (e.g., SMTP for sending and IMAP/POP3 for receiving). It should display a list of received emails and allow users to open and read them.

Managing Emails: Users should be able to manage their email inbox by marking emails as read/unread, moving emails to folders, and deleting emails.

Attachments: The email client should support sending and receiving email attachments. Users should be able to open and save attachments from received emails.

## 3. Technologies Used

* Programming Language: C
* Standard Libraries: `<stdio.h>`, `<stdlib.h>`, `<string.h>` ‘<time.h>’ `<errno.h>`,<sys/stat>’

**System Architecture**

## 4.1 Front-End

The front-end of this project is text-based and relies on user input and output through the command line.

**4.2 Back-End**

The back-end manages menu navigation, and element manipulation.

**4.3 Database**

The system does not uses any data management but stores data in file and folder

## 5. Project Modules

**Email Composition Module:**

Provides functions to create and compose new emails.

Allows users to specify recipients, subject, message body, and attach files.

Handles email formatting (e.g., text formatting options).

**Email Sending Module:**

Implements the Simple Mail Transfer Protocol (SMTP) for sending emails.

Sends composed emails to the designated recipients via a remote email server.

Handles error reporting for sending failures.

**Email Receiving Module:**

Implements either the Internet Message Access Protocol (IMAP) or the Post Office Protocol (POP3) for receiving emails.

Retrieves emails from the user's mailbox on the email server.

Parses and stores received emails locally for user access.

**Email Management Module:**

Provides functions to manage the user's email inbox.

Allows users to mark emails as read/unread, move emails to folders, and delete emails.

Manages local storage of emails and folders.

## 6. Features and Functionality

**6.1 Feature 1: Composing of email**

-We can compose the email

**6.2 Feature 2: view the composed email**

* . we can view the composed email

**6.3 Feature 3: Element Management**

* User can add elements, display, change and delete elements.

## 7. Testing

**8.1 Unit Testing**

* Each function has been tested individually.

**8.2 Integration Testing**

* The integration of functions has been tested.

**8.3 User Acceptance Testing**

* The system has been tested with user input to ensure it functions as expected.

**8. Challenges Faced**

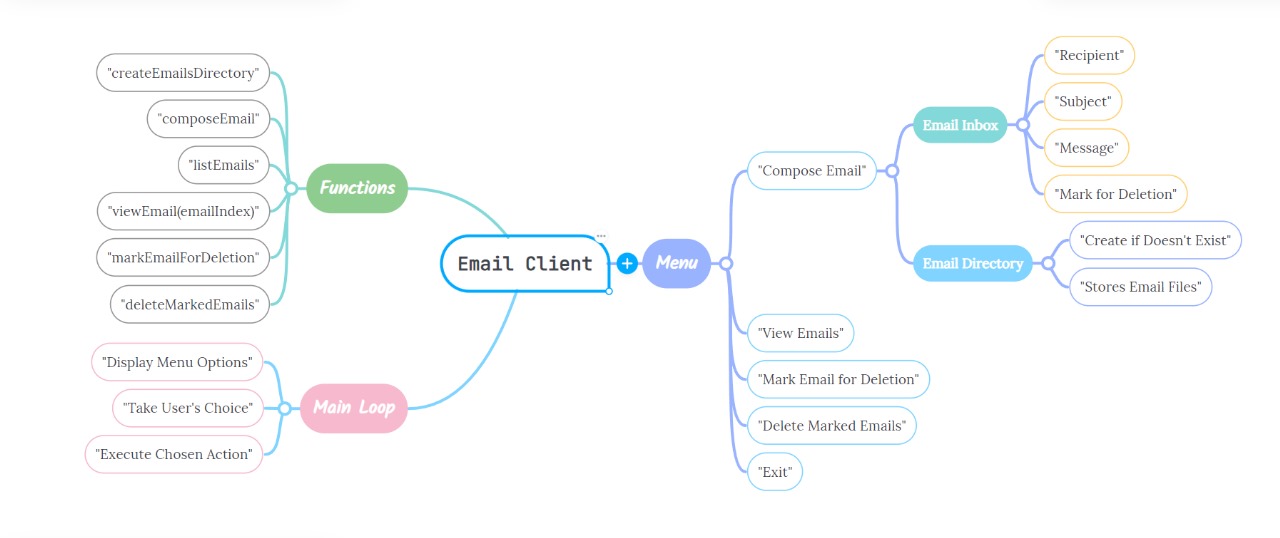
- Challenges included handling user input and implementing random algorithms.

## 9. Future Enhancements

- Future enhancements could include data persistence with file storage and a more user-friendly interface.

Time at which email is created/deleted, Integrated Task Management

**MINDMAP:**



CODE :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <errno.h>

#define MAX\_EMAILS 100 // Maximum number of emails in the inbox

#define EMAILS\_DIRECTORY "C:\\dlithe project"// Specify the email directory path

// Struct to store email details

typedef struct {

    char recipient[100];

    char subject[100];

    char message[500];

    int isDeleted; // 0 means not deleted, 1 means marked for deletion

} Email;

Email inbox[MAX\_EMAILS];

int emailCount = 0; // Track the number of emails in the inbox

// Function to create the "emails" directory if it doesn't exist

void createEmailsDirectory() {

#ifdef \_WIN32

    // Check if the directory already exists, and if not, create it

    struct stat st;

    if (stat(EMAILS\_DIRECTORY, &st) == -1) {

        if (mkdir(EMAILS\_DIRECTORY) != 0 && errno != EEXIST) {

            perror("Error creating email directory");

            exit(1);

        }

    }

#else

    // On non-Windows systems, use mkdir with permissions

    if (mkdir(EMAILS\_DIRECTORY, 0777) != 0 && errno != EEXIST) {

        perror("Error creating email directory");

        exit(1);

    }

#endif

}

// Function to compose and save an email

void composeEmail() {

    if (emailCount >= MAX\_EMAILS) {

        printf("Inbox is full. Cannot compose a new email.\n");

        return;

    }

    printf("Recipient (namenumber@gmail.com format): ");

    scanf("%s", inbox[emailCount].recipient);

    // Check if the email address is in the correct format

    if (strstr(inbox[emailCount].recipient, "@gmail.com") == NULL) {

        printf("Invalid email address format. Please use namenumber@gmail.com format.\n");

        return;

    }

    printf("Subject: ");

    scanf("%s", inbox[emailCount].subject);

    printf("Message: ");

    scanf(" %[^\n]s", inbox[emailCount].message);

    inbox[emailCount].isDeleted = 0; // Initialize as not deleted

    // Create the full path to the email file within the "emails" folder

    char filename[200]; // Adjust the size as needed

    snprintf(filename, sizeof(filename), "%s/email%d.txt", EMAILS\_DIRECTORY, emailCount);

    // Create a new file and write email details

    FILE \*fp = fopen(filename, "w");

    if (fp == NULL) {

        perror("Error creating email file");

        return;

    }

    fprintf(fp, "Recipient: %s\n", inbox[emailCount].recipient);

    fprintf(fp, "Subject: %s\n", inbox[emailCount].subject);

    fprintf(fp, "Message: %s\n", inbox[emailCount].message);

    fclose(fp);

    printf("Email has been composed and saved in the inbox.\n");

    emailCount++;

}

// Function to list emails in the inbox

void listEmails() {

    printf("Inbox:\n");

    for (int i = 0; i < emailCount; i++) {

        if (!inbox[i].isDeleted) {

            printf("%d. From: %s   Subject: %s\n", i + 1, inbox[i].recipient, inbox[i].subject);

        }

    }

}

// Function to view a specific email

void viewEmail(int emailIndex) {

    if (emailIndex < 0 || emailIndex >= emailCount || inbox[emailIndex].isDeleted) {

        printf("Invalid email selection.\n");

        return;

    }

    printf("Viewing Email %d:\n", emailIndex + 1);

    printf("Recipient: %s\n", inbox[emailIndex].recipient);

    printf("Subject: %s\n", inbox[emailIndex].subject);

    printf("Message: %s\n", inbox[emailIndex].message);

    int option;

    printf("\nOptions:\n");

    printf("1. Reply to this email\n");

    printf("2. Forward this email\n");

    printf("3. Back to Inbox\n");

    printf("Enter your choice: ");

    scanf("%d", &option);

    switch (option) {

        case 1:

            replyToEmail(emailIndex);

            break;

        case 2:

            forwardEmail(emailIndex);

            break;

        case 3:

            // Do nothing, return to the inbox

            break;

        default:

            printf("Invalid option. Returning to the inbox.\n");

    }

}

// Function to reply to an email

void replyToEmail(int emailIndex) {

    if (emailCount >= MAX\_EMAILS) {

        printf("Inbox is full. Cannot compose a reply.\n");

        return;

    }

    printf("Replying to Email %d:\n", emailIndex + 1);

    printf("Recipient (namenumber@gmail.com format): ");

    scanf("%s", inbox[emailCount].recipient);

    // Check if the email address is in the correct format

    if (strstr(inbox[emailCount].recipient, "@gmail.com") == NULL) {

        printf("Invalid email address format. Please use namenumber@gmail.com format.\n");

        return;

    }

    // Set the subject and message for the reply

    snprintf(inbox[emailCount].subject, sizeof(inbox[emailCount].subject), "Re: %s", inbox[emailIndex].subject);

    printf("Subject: %s\n", inbox[emailCount].subject);

    printf("Message: ");

    scanf(" %[^\n]s", inbox[emailCount].message);

    inbox[emailCount].isDeleted = 0; // Initialize as not deleted

    // Create the full path to the email file within the "emails" folder

    char filename[200]; // Adjust the size as needed

    snprintf(filename, sizeof(filename), "%s/email%d.txt", EMAILS\_DIRECTORY, emailCount);

    // Create a new file and write email details

    FILE \*fp = fopen(filename, "w");

    if (fp == NULL) {

        perror("Error creating email file");

        return;

    }

    fprintf(fp, "Recipient: %s\n", inbox[emailCount].recipient);

    fprintf(fp, "Subject: %s\n", inbox[emailCount].subject);

    fprintf(fp, "Message: %s\n", inbox[emailCount].message);

    fclose(fp);

    printf("Reply email has been composed and saved in the inbox.\n");

    emailCount++;

}

// Function to forward an email

void forwardEmail(int emailIndex) {

    if (emailCount >= MAX\_EMAILS) {

        printf("Inbox is full. Cannot forward this email.\n");

        return;

    }

    printf("Forwarding Email %d:\n", emailIndex + 1);

    printf("Recipient (namenumber@gmail.com format): ");

    scanf("%s", inbox[emailCount].recipient);

    // Check if the email address is in the correct format

    if (strstr(inbox[emailCount].recipient, "@gmail.com") == NULL) {

        printf("Invalid email address format. Please use namenumber@gmail.com format.\n");

        return;

    }

    // Set the subject and message for the forwarded email

    snprintf(inbox[emailCount].subject, sizeof(inbox[emailCount].subject), "Fwd: %s", inbox[emailIndex].subject);

    printf("Subject: %s\n", inbox[emailCount].subject);

    printf("Message: ");

    scanf(" %[^\n]s", inbox[emailCount].message);

    inbox[emailCount].isDeleted = 0; // Initialize as not deleted

    // Create the full path to the email file within the "emails" folder

    char filename[200]; // Adjust the size as needed

    snprintf(filename, sizeof(filename), "%s/email%d.txt", EMAILS\_DIRECTORY, emailCount);

    // Create a new file and write email details

    FILE \*fp = fopen(filename, "w");

    if (fp == NULL) {

        perror("Error creating email file");

        return;

    }

    fprintf(fp, "Recipient: %s\n", inbox[emailCount].recipient);

    fprintf(fp, "Subject: %s\n", inbox[emailCount].subject);

    fprintf(fp, "Message: %s\n", inbox[emailCount].message);

    fclose(fp);

    printf("Forwarded email has been composed and saved in the inbox.\n");

    emailCount++;

}

// Function to mark an email for deletion

void markEmailForDeletion() {

    int emailIndex;

    printf("Enter the number of the email you want to mark for deletion: ");

    scanf("%d", &emailIndex);

    if (emailIndex > 0 && emailIndex <= emailCount) {

        inbox[emailIndex - 1].isDeleted = 1;

        printf("Email marked for deletion.\n");

    } else {

        printf("Invalid email selection.\n");

    }

}

// Function to permanently delete marked emails

void deleteMarkedEmails() {

    int markedCount = 0;

    for (int i = 0; i < emailCount; i++) {

        if (inbox[i].isDeleted) {

            markedCount++;

        } else if (markedCount > 0) {

            // Shift remaining emails to fill the gaps

            inbox[i - markedCount] = inbox[i];

        }

    }

    emailCount -= markedCount;

    printf("%d email(s) marked for deletion have been permanently deleted.\n", markedCount);

}

int main() {

    createEmailsDirectory(); // Create the "emails" directory if it doesn't exist

    int choice;

    while (1) {

        printf("\nSimple Email Client Menu:\n");

        printf("1. Compose Email\n");

        printf("2. View Emails\n");

        printf("3. Mark Email for Deletion\n");

        printf("4. Delete Marked Emails\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                composeEmail();

                break;

            case 2:

                listEmails();

                break;

            case 3:

                markEmailForDeletion();

                break;

            case 4:

                deleteMarkedEmails();

                break;

            case 5:

                printf("Exiting the email client. Goodbye!\n");

                exit(0);

            default:

                printf("Invalid choice. Please select a valid option.\n");

        }

    }

    return 0;

}

Output:

Simple Email Client Menu:

1. Compose Email

2. View Emails

3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 1

Recipient (namenumber@gmail.com format): srijan2@gmail.com

Subject: Greetings

Message: hello Sir

Email has been composed and saved in the inbox.

Simple Email Client Menu:

1. Compose Email

2. View Emails

3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 1

Recipient (namenumber@gmail.com format): Mahesh234@gmail.com

Subject: Reply

Message: How are you?

Email has been composed and saved in the inbox.

Simple Email Client Menu:

1. Compose Email

2. View Emails

3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 2

Inbox:

1. From: srijan2@gmail.com Subject: Greetings

2. From: Mahesh234@gmail.com Subject: Reply

Simple Email Client Menu:

1. Compose Email

2. View Emails

3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 3

Enter the number of the email you want to mark for deletion: 2

Email marked for deletion.

Simple Email Client Menu:

1. Compose Email

2. View Emails

3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 4

1 email(s) marked for deletion have been permanently deleted.

Simple Email Client Menu:

1. Compose Email

2. View Emails

3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 2

Inbox:

1. From: srijan2@gmail.com Subject: Greetings

Simple Email Client Menu:

1. Compose Email

2. View Emails

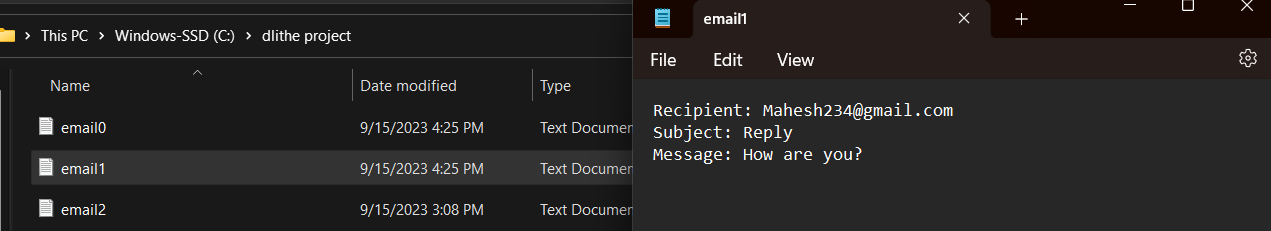
3. Mark Email for Deletion

4. Delete Marked Emails

5. Exit

Enter your choice: 5

Exiting the email client. Goodbye!



**In conclusion:**

In conclusion, creating a basic email client in C is a challenging but highly educational project that allows developers to apply their skills to real-world problems. It offers insights into both the technical and user experience aspects of software development. Additionally, it can serve as a foundation for more advanced email client projects or integration into larger systems.

**REFERENCES:**

1. C Programming Books:

• "C Programming Absolute Beginner's Guide" by Perry and Miller

• "C Programming for the Absolute Beginner, Second Edition" by Vine

• "C Programming for the Absolute Beginner" by Vine and Vine

2. Online C Programming Tutorials:

• C Programming - GeeksforGeeks

• C Programming Tutorial - Tutorialspoint

• C Programming - Learn-C.org

3. C Language Documentation:

• C Standard Library Reference - cppreference.com

4. Data Structures and Algorithms:

• "Data Structures and Algorithms in C" by Drozdek

• "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein

5. Random Number Generation:

• Random Number Generation in C - GeeksforGeeks

• rand() function - cppreference.com

6. Menu-Driven Programs:

• Creating a Simple Menu in C - GeeksforGeeks

• How to Create Menus in C - Tutorialspoin