Plagiarism Scan Report By SmallSEOTools

Report Generated on: Oct 29,2024

Plagiarized Content

0% Exact Plagiarized

0% Partial Plagiarized

Unique Content

Total Words: 685 **Total Characters:** 4315 **Plagiarized Sentences:** 0 **Unique Sentences:** 8 (100%)

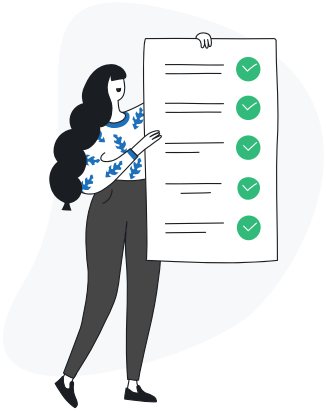
Content Checked for Plagiarism

```
#include
#include
using namespace std;

// Class to store the email details with private data members
class Email {
private:
string sender;
string receiver;
string subject;
string body;

public:
Email* next; // Pointer to the next email (acting as a node)

// Default constructor
Email() : next(nullptr) {}

// Parameterized constructor
Email(string s, string r, string sub, string b) {
sender = s;
receiver = r;
subject = sub;
body = b;
next = nullptr;
}

// Getters to access private data members
string getSender() const { // Marked as const
return sender;
}
string getReceiver() const { // Marked as const
return receiver;
}
string getSubject() const { // Marked as const
return subject;
```

```
}
string getBody() const { // Marked as const
return body;
}

// Display email details
void display() {
cout << "From: " << sender << ", To: " << receiver
<< ", Subject: " << subject << ", Body: " << body << endl;
}
};

// Class to manage the email queue using a linked list
class EmailQueue {
private:
Email* front; // Pointer to the front of the queue
Email* rear; // Pointer to the rear of the queue

public:
// Constructor to initialize an empty queue
EmailQueue() : front(nullptr), rear(nullptr) {}

// Function to enqueue (add) an email to the queue
void enqueue(const Email& email) {
Email* newNode = new Email(email); // Create a new node dynamically
if (rear == nullptr) { // If queue is empty
front = rear = newNode;
} else { // Else add new node at the rear
rear->next = newNode; // Use public next pointer
rear = newNode;
}
cout << "Email from " << email.getSender() << " to " << email.getReceiver()
<< " has been added to the queue." << endl;
}

// Function to dequeue (remove) an email from the front of the queue
void dequeue() {
if (front == nullptr) { // If the queue is empty
cout << "Queue is empty." << endl;
return;
}

Email* temp = front; // Store the front email
front = front->next; // Move the front pointer to the next email

if (front == nullptr) { // If the queue becomes empty
rear = nullptr;
}

cout << "Email from " << temp->getSender() << " to " << temp->getReceiver()
<< " has been processed and removed from the queue." << endl;

delete temp; // Free the memory of the dequeued node
}

// Function to display all emails in the queue
void displayAllEmails() {
if (front == nullptr) {
cout << "No emails in the queue." << endl;
```

```
return;
}

Email* temp = front;
int index = 1;
cout << "Emails in the queue:" << endl;
while (temp != nullptr) { // Traverse through the linked list
cout << index++ << ". ";
temp->display();
temp = temp->next; // Use public next pointer
}
}

// Destructor to free all dynamically allocated memory
~EmailQueue() {
while (front != nullptr) {
Email* temp = front;
front = front->next; // Use public next pointer
delete temp;
}
}
};

// Function to display the menu options
void displayMenu() {
cout << "\nMenu:\n";
cout << "1. Enqueue Email\n";
cout << "2. Dequeue Email\n";
cout << "3. Display All Emails\n";
cout << "4. Exit\n";
cout << "Enter your choice: ";
}

// Main function to drive the email queue management
int main() {
EmailQueue emailQueue;
int choice;

while (true) {
displayMenu();
cin >> choice;

switch (choice) {
case 1: {
// Enqueue email
string sender, receiver, subject, body;
cout << "Enter sender email: ";
cin >> sender;
cout << "Enter receiver email: ";
cin >> receiver;
cout << "Enter subject: ";
cin.ignore(); // To ignore leftover newline character
getline(cin, subject);
cout << "Enter body: ";
getline(cin, body);

Email email(sender, receiver, subject, body);
emailQueue.enqueue(email);
break;
```

```
}
case 2: {
// Dequeue email
emailQueue.dequeue();
break;
}
case 3: {
// Display all emails
emailQueue.displayAllEmails();
break;
}
case 4: {
cout << "Exiting the program." << endl;
return 0;
}
default: {
cout << "Invalid choice, please try again." << endl;
}
}
}

return 0;
}
```



No Plagiarism Found