Name: Timaz Amir

Roll No: 24L-0706

Class:BCS-2K

Title: Social Interaction System

Social Interaction System

Abstract

The Social Interaction System is a comprehensive C++ application that simulates a social media platform where users can create profiles, connect with friends, share posts, join groups, follow pages, and exchange messages. This console-based system implements core object-oriented programming concepts including inheritance, polymorphism, encapsulation, and dynamic memory management to create a robust social network experience. The application stores user data persistently through file serialization, allowing users to maintain their social connections across sessions.

Table of Contents

Introduction

Problem Statement

In today's digitally connected world, social interaction platforms serve as essential tools for human connection and information sharing. However, developing a comprehensive system that manages user profiles, content sharing, and interpersonal communications presents significant technical challenges, particularly in memory management, data relationships, and system organization.

Objectives

- Create a modular, object-oriented social media application that demonstrates proper C++ programming practices
- Implement a persistent user data system through file serialization
- Design a comprehensive social network that includes user profiles, content sharing, messaging, groups, and pages
- Incorporate robust validation for user inputs and system behaviors
- Demonstrate advanced OOP concepts including inheritance, polymorphism, and encapsulation

Motivations

This project was motivated by the desire to explore complex object relationships in a real-world application context. Social networks represent intricate systems where entities (users, posts, groups) maintain numerous interdependent relationships. By implementing such a system, the project demonstrates how OOP principles can elegantly solve complex data management problems while providing an engaging user experience.

The system also serves as an educational platform for understanding key programming concepts such as dynamic memory allocation, data serialization, user authentication, and input validation—all critical components in modern software development.

OOP Concepts Used

Inheritance

The system extensively utilizes inheritance to establish hierarchical relationships between related classes, promoting code reuse and logical organization:

1. Profile Hierarchy:
   - The base `Profile` class defines common attributes and behaviors for all profile types
   - `PersonalProfile` and `BusinessProfile` extend `Profile` with specialized attributes and behaviors
   - This inheritance structure allows for treating different profile types polymorphically while enabling specialized behaviors

2. Message Hierarchy:
   - `Message` serves as the base class for communication functionality
   - `GroupMessage` extends `Message` with group-specific features
   - This inheritance structure eliminates code duplication while allowing for different message handling workflows

Polymorphism

Polymorphism enables the system to process different object types through a common interface:

1. Profile Management:

   - Virtual functions like `readProfile()` allow different profile types to be handled uniformly

   - This enables the system to display and manage diverse profile types without type-specific code branches

2. Timeline Content:

   - Different post types can be displayed and interacted with through common interfaces

   - This simplifies timeline management code while supporting diverse content types

Encapsulation

Encapsulation protects data integrity by hiding implementation details:

1. User Data Protection:

   - Private member variables for sensitive data (password, email, etc.)

   - Public accessor methods that enforce validation rules

   - This prevents direct manipulation of critical user data

2. Post Management:

   - Internal post data structures are encapsulated

   - Interactions are mediated through well-defined methods like `PostMaker()`, `LikeIncreaser()`, etc.

   - This ensures post interactions follow proper validation and business rules

Composition and Aggregation

Complex relationships between objects are managed through composition and aggregation:

1. User-Profile Relationship (Composition):
   - A `User` contains a `Profile` as a core component
   - The Profile's lifecycle is tied to the User (composition)
   - This ensures profile data remains consistent with user data

2. User-Friends Relationship (Aggregation):
   - Users maintain collections of other User objects as friends
   - These relationships use pointers/references, forming an aggregation
   - This allows flexible social connections between independent entities

3. Post-Comment Relationship (Composition):
   - Posts own and manage their comments
   - When a post is deleted, its comments are also removed
   - This preserves data integrity in content interactions

Test Cases

User Registration and Authentication

Input:

1. Select option 2 (SignUp)
2. Enter username: "user123"
3. Enter password: "P@ssw0rd"
4. Enter email: "user123@example.com"
5. Enter birthdate: "01-01-1990"

6. Log out and select option 1 (Login)

7. Enter username: "user123"

8. Enter password: "P@ssw0rd"

Expected Output:

- Message: "The User-name must be at least 5 characters containing number and alphabet"

- Password accepted (meets all criteria)

- Email accepted

- Birthdate accepted

- User registration successful

- Login successful

- Main interface menu displayed

Profile Customization

Input:

1. Login as "user123"

2. Select option 1 (Customize Profile)

3. Select option 2 (Personal Information)

4. Enter last name: "Smith"

5. Enter phone number: "1234567890"

6. Enter location: "New York"

7. Enter student status: "1" (Yes)

8. Enter organization: "University XYZ"

Expected Output:

- Profile information updated successfully
- Updated profile data saved to system

Post Creation and Interaction

Input:

1. Login as "user123"
2. Select option 4 (Create Post)
3. Enter post content: "Hello world! This is my first post."
4. Select option 5 (View My TimeLine)
5. Select post ID: 1
6. Select option 1 (Like)
7. Return to timeline, select post ID: 1
8. Select option 2 (Comment)
9. Enter comment: "This is a comment on my own post"

Expected Output:

- Post uploaded successfully
- Timeline displays the post with content "Hello world! This is my first post."
- Post liked successfully
- Comment posted successfully
- Timeline shows updated post with 1 like and 1 comment

Friend Request and Management

Input:

1. Create another user "friend456" with valid credentials
2. Login as "user123"

3. Select option 2 (Search for Users)

4. Enter username: "friend456"

5. Select option 1 (Add Friend)

6. Log out and login as "friend456"

7. Select option 3 (View Your Friend Request Box)

8. Select request 1 (from "user123")

9. Select option 1 (Accept)

Expected Output:

- "User Found" message displayed

- "Friend Request send successful" message displayed

- Display of pending friend request from "user123"

- "Friend Added successfully" message displayed

- Both users now show each other in their friends list

Group Creation and Joining

Input:

1. Login as "user123"

2. Select option 6 (Make a new Group)

3. Enter group name: "Programming Enthusiasts"

4. Log out and login as "friend456"

5. Select option 7 (View all groups and join one)

6. Select option 1 (Yes) to join a group

7. Select group 1 (Programming Enthusiasts)

Expected Output:

- "Group created successfully" message displayed

- Group listed with name "Programming Enthusiasts" and 1 member

- "Group joined successfully" message displayed

- Group now shows 2 members

Page Creation and Following

Input:

1. Login as "user123"

2. Select option 8 (Make a Page)

3. Enter page name: "Tech News"

4. Log out and login as "friend456"

5. Select option 9 (View all Pages and follow one)

6. Select option 1 (Yes) to follow a page

7. Select page 1 (Tech News)

Expected Output:

- "Page created successfully" message displayed

- Page listed with name "Tech News" and 1 follower

- "You have successfully followed this page" message displayed

- Page now shows 2 followers

Messaging Between Users

Input:

1. Login as "user123"

2. Select option 2 (Search for Users)

3. Enter username: "friend456"

4. Select option 3 (Send Message)

5. Enter message: "Hello friend456, how are you?"

6. Log out and login as "friend456"

7. Select option 2 (Search for Users)

8. Enter username: "user123"

9. Select option 2 (View Messages)

10. Select option 1 (Yes) to reply

11. Enter message: "I'm doing well, thanks for asking!"

Expected Output:

- "Message sent successful" message displayed
- Messages displayed while viewing as "friend456"
- "Reply sent successful" message displayed
- Both users can see the complete conversation

Future Directions

Enhanced User Experience

1. Graphical User Interface: Implement a GUI using frameworks like Qt or SFML to improve visual appeal and usability

2. Real-time Notifications: Add a notification system for friend requests, messages, and post interactions

3. Multi-media Content Support: Extend post functionality to include photos, videos, and other media types

4. User Tagging: Enable tagging friends in posts and comments with special formatting and notifications

Advanced Social Features

1. Privacy Controls: Implement granular privacy settings for posts and profile information

2. Content Feed Algorithms: Develop algorithms to personalize content based on user interests and interactions

3. Event Management: Add event creation, invitation, and RSVP functionalities

4. Location-based Services: Integrate location features for check-ins and proximity-based friend suggestions

5. Post Sharing: Allow users to share posts from other users or pages to their own timeline

Technical Improvements

1. Database Integration: Replace file-based storage with a SQL or NoSQL database for better performance and scalability

2. Client-Server Architecture: Restructure as a client-server application to enable multi-user access

3. API Development: Create a RESTful API that allows third-party integration

4. Security Enhancements: Improve password hashing, add two-factor authentication, and implement session management

5. Concurrency Support: Add multi-threading to handle simultaneous user operations

Analytics and Insights

1. User Activity Metrics: Generate reports on user engagement, content popularity, and interaction patterns

2. Network Analysis: Provide insights into social connections and community structures

3. Content Recommendations: Suggest friends, groups, and pages based on user behavior and preferences

4. Trending Topics: Identify and highlight trending discussions across the platform

Business Features

1. Advertising System: Implement targeted advertising capabilities for business accounts

2. Marketplace: Add functionality for users to buy, sell, or exchange items

3. Business Analytics: Provide engagement metrics for business pages and promotional posts

4. Verified Accounts: Implement verification processes for public figures and organizations

5. Subscription Services: Add premium features accessible through subscription models

UML DIAGRAM:

Social Interaction System - UML Class Diagram

User Class

```
+----------------------------------+
|           User             |
+----------------------------------+
| - userID: int                |
| - name: string               |
| - email: string              |
| - password: string           |
| - birthdate: string          |
| - YourMessages: Message      |
| - MyProfile: Profile         |
| - PicPath: string            |
|  ConfirmedFriends: User      |
|  FriendCount: int            |
|  FriendMaxCount: int         |
|  FriendRequests: User        |
|  RequestCount: int           |
|  MaxCount: int               |
+----------------------------------+
| + User()                     |
| + User(int, string...)       |
| + serialize(): void          |
| + deserialize(): void        |
| + PostMaker(): void          |
| + TimeLinerPrinterOwn(): void    |
| + MessageViewer(): bool          |
| + MessageMaker(): void           |
| + TimeLinePrinterVisitor(): void |
| + AddFriendRequest(): void       |
| + AddFriend(): void              |
| + ShowFriendRequest(): void      |
| + ProfileCustomizer(): void      |
+----------------------------------+
```

Profile Class Hierarchy

```
+----------------------------------+
|          Profile             |
+----------------------------------+
| - firstName: string            |
| - lastName: string             |
| - dob: string                  |
| - phoneNumber: string          |
| - email: string                |
| - location: string             |
| - isStudent: bool              |
| - organization: string         |
|  MyPosts: Post                 |
|  MaxPost: int                  |
|  numPosts: int                 |
+----------------------------------+
| + ProfileCustomizer(): void     |
| + PostUploader(): void         |
| + MyTimeLineDisplayer(): bool   |
| + MyPostSelector(): int        |
| + MyPostInteractionLike(): void |
| + readProfile(): virtual void   |
+----------------------------------+
              |
   +-----------+-----------+
   |                   |
+------------------+  +----------------------+
| PersonalProfile  | |   BusinessProfile    |
+------------------+  +----------------------+
| + hobbies: string |  | + companyName: string |
+------------------+  | + NumberOfPeople: int |
| + readProfile()  |  +----------------------+
+------------------+  | + readProfile()      |
                  +----------------------+
```

Post Class

```
+----------------------------------+
|            Post              |
+----------------------------------+
| - postID: int                  |
| - content: string              |
| - timestamp: time_t            |
| - PostLikes: Like             |
| - PostComments: Comment       |
| - LikeCount: int              |
| - MaxLikeCount: int           |
| - CommentCount: int           |
| - MaxCommentCount: int        |
+----------------------------------+
| + Post()                      |
| + Post(int, string)           |
| + LikeIncreaser(): void       |
| + CommmentIncrease(): void    |
| + getPostID(): int            |
| + getContent(): string        |
| + getTimeStamp(): time_t      |
| + PostDisplayer(): void       |
| + setPostID(): void           |
| + setContent(): void          |
+----------------------------------+
```

Comment Class

```
+----------------------------------+
|          Comment              |
+----------------------------------+
| - content: string             |
+----------------------------------+
| + CommentIncrease(): void     |
| + getContent(): string         |
| + setContent(): void           |
+----------------------------------+
```

Like Class

```
+----------------------------------+
|            Like            |
+----------------------------------+
| - count: int                  |
+----------------------------------+
| + LikeIncreaser(): void        |
+----------------------------------+
```

Message Class Hierarchy

```
+----------------------------------+
|          Message           |
+----------------------------------+
| - messageID: int              |
| - sender: Messager            |
| - content: string             |
| - timestamp: time_t           |
+----------------------------------+
| + Message()                   |
| + Message(int, string)         |
| + serialize(): void           |
| + deserialize(): void          |
| + getMessageID(): int          |
| + getSender(): Messager        |
| + getContent(): string         |
| + getTimestamp(): time_t       |
| + DisplayMessages(): void      |
+----------------------------------+
             |
+----------------------------------+
|        GroupMessage          |
+----------------------------------+
| - GroupMessagers: Messager     |
+----------------------------------+
| + GroupMessage()               |
+----------------------------------+
```

Group Class

```
+----------------------------------+
|           Group            |
+----------------------------------+
| + GroupName: string            |
| + MyGroupMembers: GroupMembers  |
| + MemberCount: int             |
| + MaxMemberCount: int          |
+----------------------------------+
| + Group()                  |
| + serialize(): void            |
| + deserialize(): void          |
| + AddMember(): bool            |
+----------------------------------+
```

Page Class

```
+----------------------------------+
|            Page            |
+----------------------------------+
| - MyFollowers: Follower        |
| - FollowerCount: int           |
| - MaxFollowerCount: int        |
| + PageName: string             |
+----------------------------------+
| + serialize(): void            |
| + deserialize(): void          |
| + AddFollower(): bool          |
| + PageDisplayer(): void        |
+----------------------------------+
```

InteractionSystem Class

```
+----------------------------------+
|      InteractionSystem        |
+----------------------------------+
| - UserCount: int               |
| - MaxUser: int                 |
| - MyUsers: User                |
| - MyGroups: Group            |
| - GroupCount: int            |
| - MaxGroupCount: int          |
| - MyPages: Page              |
| - PageCount: int             |
| - PageMaxCount: int           |
+---------------------------------+
| + serialize(): void           |
| + deserialize(): void         |
| + readObjectFromFile(): IS      |
| + writeObjectToFile(): void     |
| + EmailVerifier(): bool        |
| + PasswordVerifier(): bool      |
| + SearchUser(): void          |
| + LoginVerifier(): bool        |
| + GroupViewer(): void         |
| + GroupMaker(): void          |
| + PageMaker(): void           |
| + PageViewer(): void          |
| + InterfaceMenue(): void       |
| + LoginMaker(): void          |
| + SignUpMaker(): void         |
| + MainMenue(): void           |
+---------------------------------+
```

Validator Class

```
+----------------------------------+
|        Validator           |
+----------------------------------+
| + MyValidator(): bool          |
| + UserNameVerifier(): bool      |
+----------------------------------+
```

Relationships:

1. User has-a Profile (Composition)

2. Profile has-many Posts (Composition)

3. Post has-many Comments (Composition)

4. Post has-many Likes (Composition)

5. User has-many Friends (User objects - Aggregation)

6. User has-many Messages (Composition)

7. PersonalProfile is-a Profile (Inheritance)

8. BusinessProfile is-a Profile (Inheritance)

9. GroupMessage is-a Message (Inheritance)

10. InteractionSystem manages all other entities