Seth Denney - std0003
COMP 2710 Lab 1
Program Design Outline

I. Use Cases

A. Case – User Views Friend Group's Messages

i. User logs in. User chooses to display his or her home page. Aubiebook displays the last three messages from the user's friend group. If more than three messages are available, Aubiebook displays an option to show all messages. The user chooses this option. All messages from the user's friend group are displayed in reverse chronological order. The user then quits Aubiebook.

B. Case – User Types a Message and Views Own Message History

i. User logs in. User chooses to post a message. After posting the message, the user chooses to display his or her wall page. Aubiebook displays the last three messages posted by the user. If more than three messages are available, Aubiebook displays an option to show all messages. The user chooses this option. All messages posted by the user are displayed in reverse chronological order. The user then quits Aubiebook.

C. Case – User Creates Profile, Switches Users, and Adds a Friend

i. User chooses to create a new user, and enters a user name. The user then switches to an existing user profile, and chooses to add the newly created user as a friend on Aubiebook. After selecting the 'Add a Friend' option, the user enters the desired user name, and the appropriate profile is added to the current user's friend list. The user then quits Aubiebook.

II. Classes

A. **AubieBook** – The *Aubiebook* class holds all system-level functionality in the Aubiebook application. It contains a *main()* method that will be run as the primary driver function for Aubiebook. It uses the *User* class and the *Page* class to store message data and perform object-specific functions.

i. Methods

a) void *createUser()*
- instantiate a *User* object
- add *User* object to *validUsers*
- prints welcome banner
- sets *currentUser* to new User object

b) void *displayMenu()*
- display main menu options
- read in user menu selection
- call appropriate function (switch statement)

c) void *postMessage()*
- prompt for message
- read in message ('$\n' terminated)
- call *page.addMessage(*message*)* (sans '$')

d) void *displayWall()*
- call *page.displayPage(currentUser, "wall")*

e) void *displayHome()*
- *call page.displayPage(currentUser, "home")*

f) void *addFriend()*
- prompt for name of friend
- read in name
- check name against *validUsers*; if valid, call *currentUser.addFriend(name)*; if not valid, print error message
- call *currentUser.addFriend(name)*

g) void *switchUser()*
- prompt for name of next user
- read in name
- check name against *validUsers*; if valid, set *currentUser* to next user; if not valid, print error message and return to top of function (while loop)
- after successful user switch, print Welcome Back Banner
- call *displayMenu()*

h) void *quitApp()*
- exit gracefully

i) int *findUser(name)*
- returns the index of the desired User object where *user.getName() == name*

ii. Data Fields

a) User *currentUser*
- pointer to currently selected *User* object

b) vector<User> *validUsers*
- vector that includes a pointer to every previously created *User* object

c) Page *page*
   - Page object that stores and prints messages

d) bool *loggedIn*
   - boolean that is set to true only after the first user has logged in

B. User – The *User* class represents a single user profile on Aubiebook. It stores the user's user name and friends list, as well as functions to interact with this data. The *User* class is used by both *Aubiebook* and *Page*.

i. Methods

a) void *addFriend(*string *friend)*
   - adds *friend* to *friendsList* if not already found in *friendsList*

b) boolean *isFriend(*string *friend)*
   - return true if *friend* is member of *friendsList*

c) string *getName()*
   - return *name*

ii. Data Fields

a) string *name*
   - the user name

b) vector<string> *friendsList*
   - vector that includes a string username for each of the user's friends

C. Page – The *Page* class stores data and functionality related to page-viewing. The *messageBuffer* string is stored within an instance of the *Page* class via the *Aubiebook* class. *Page* is used by the *Aubiebook* class.

i. Methods

a) void display*Page(*User *user*, string *pageType)*
   - print appropriate title banner (wall or home)
   - if *pageType* is *"wall"*, print only messages where the username is *user.name*
     - display latest 3 (formatted) messages from current user in reverse chronological order
     - if more messages exist, print prompt to display remaining messages
     - if three or fewer messages exist, do not prompt
   - if *pageType* is *"home"*, print only messages where *user.isFriend(*username*)* is 'true'
     - display latest 3 (formatted) messages from current user's friends list in reverse chronological order
     - if more messages exist, print prompt to display remaining messages
     - if three or fewer messages exist, do not prompt

b) void *addMessage(*string *name*, string *message)*
   - append formatted *name* and *message* to front of *messageBuffer*

c) void *printMessages(*User& user, string *pageType*, int limit*)*
   - iterates through *messageBuffer* and selectively prints up to <*limit*> messages to the screen, as designated by *user*, and *pageType*

ii. Data Fields

a) string *messageBuffer*
   - contains a list of buffered usernames and messages in reverse chronological order

III. Test Cases

A. First User Invalid Menu Options

i. Scenario – First user opens application and selects 'Post a message', 'Display wall page', 'Display home page', 'Add a friend', or 'Switch to a different user' .

ii. Result – System should print an error message and return to menu view. If invalid option is selected repeatedly, repeat result until a valid option is selected.

B. Second User Valid Menu Options

i. Scenario – Second user logs into already open application and selects 'Post a message', 'Display wall page', 'Display home page', 'Add a friend', or 'Switch to a different user'.

ii. Result – System should perform selected operation and return to menu.

C. Invalid Command

i. Scenario – User enters a command (menu option ID number, yes/no, user name) that is either a) incorrectly typed data (integer instead of lettered name, non-integer characters for an integer menu option, etc.) or b) correctly typed, but not within the domain of acceptable options ("100" where only "1" - "7" would be allowed, or the name of a user that has not yet created a profile)

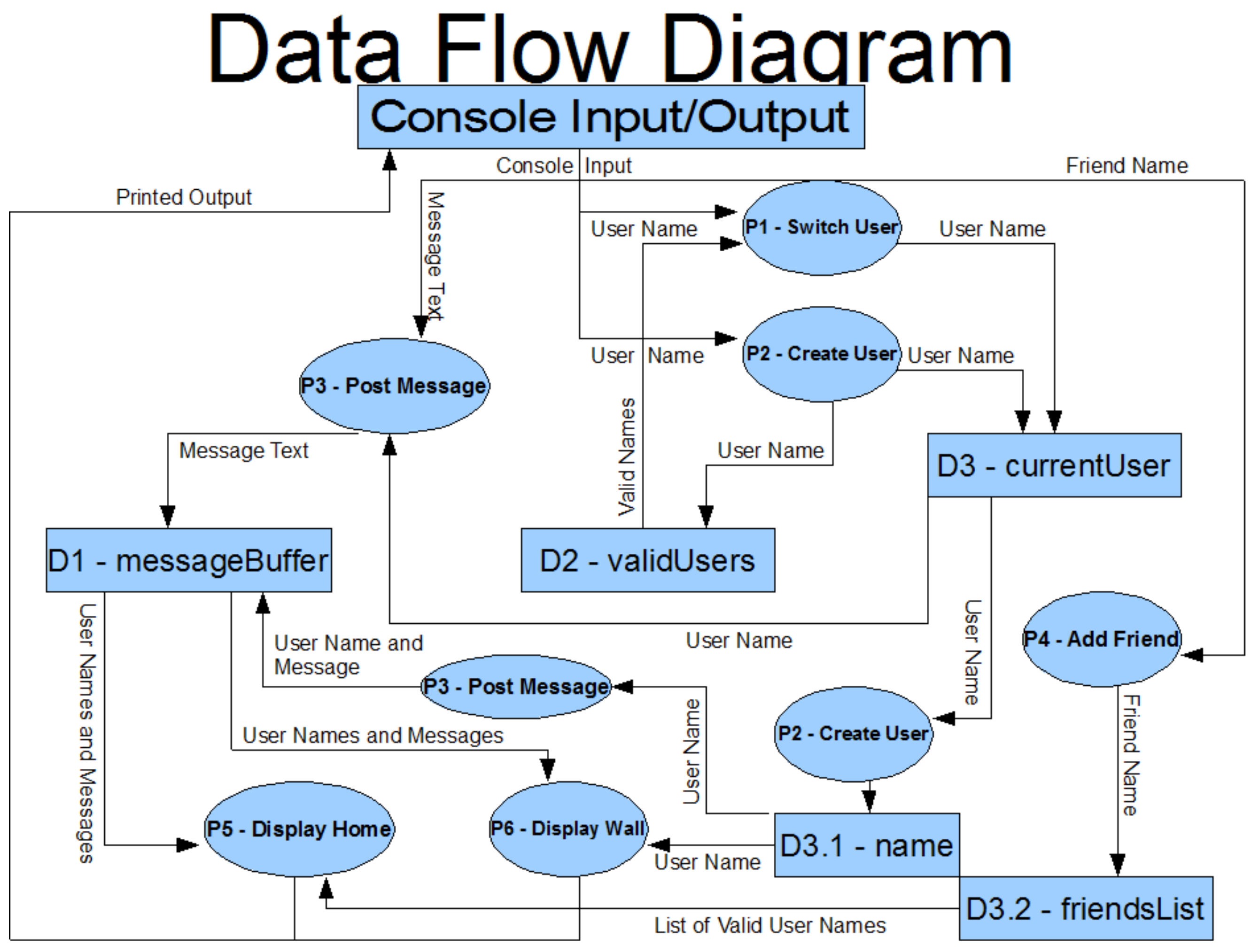
ii. Result – System should print an error message describing the nature of the error, and either a) prompt for a correct entry, or b) return to the menu view.

D. Null Parameter

i. Scenario – A message, name, or other parameter is passed to a function within the Aubiebook implementation is *null*.

ii. Result – Receiving function should handle the null parameter and not cause a runtime error to occur. If *null* was passed because of a user error, Aubiebook should print an error message to the screen detailing the issue, and offering advice on how to avoid such errors in the future.

IV. Data Flow Diagram


Data Flow Diagram
Console Input/Output
Printed Output
Console Input
Friend Name
Message Text
User Name
P1 - Switch User
User Name
User Name
P2 - Create User
User Name
Valid Names
P3 - Post Message
Message Text
User Name
D3 - currentUser
D1 - messageBuffer
D2 - validUsers
User Names and Messages
User Name and Message
User Name
User Name
P4 - Add Friend
P3 - Post Message
User Name
P2 - Create User
Friend Name
User Names and Messages
P5 - Display Home
P6 - Display Wall
D3.1 - name
User Name
D3.2 - friendsList
List of Valid User Names