ISSN NO. 2320-5407

Journal homepage: http://www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH

RESEARCH ARTICLE

Phase book Simulator with No Back-end

Dr. S. Satyanarayana[1], P. Sravan Kumar[2], V. Tata Rao[3]

1. Professor, Dept. of CSE Raghu Engineering College Visakhapatnam, Andhra Pradesh, INDIA
2. Associate Professor: Dept. of CSE Raghu Engineering College Visakhapatnam, Andhra Pradesh, INDIA
3. Assistant Professor, Dept of CSE Raghu Engineering College Visakhapatnam, Andhra Pradesh, INDIA

Manuscript Info

Manuscript History:

Received: 18 September 2015
Final Accepted: 22 October 2015
Published Online: November 2015

Key words:

linked list; class; objects; node; allocation; deallocation

Corresponding Author

Dr. S. Satyanarayana

Abstract

This paper proposes the algorithms that prove the strength of data structures. To prove this we have taken an example simulator to Facebook without any back end database management system

INTRODUCTION

Usually when a user wants to work with any social networking. There will various operations the members will do:

a) A new member joins

b) A current user quits

c) Member A sends a friend request to user B

d) Member B accepts a friend request from user A

e) Member A un-friends B

f) Print member A's account information, including:

- all friends of A
- all pending friend request sent by member A
- all pending friend request received by member A

g) List all members and all their friends

- PROPOSED SIMULATION SYSTEM

- *Member's name*

This is primary attribute of the user account. Possibility of this is a unique mail id or user name. As per the

implementation it can be a string value.

- *List of friends*

This is a linear data list contains friends attached to current member. It starts with a header node and traverse up to null. Null represents end of friends list or no friends are available to current member.

C. List of pending friend request sent

It is a also a linear list contains friends requests sent by current member to other members of this social networking. No requests pending is represented by NULL.

D. List of pending friend request received

A linear list of members that who are requested current members to become friend. It is also ended with NULL or no friends request is also shows NULL.

- OPERATIONAL COMMANDS

All this simulator works on single letter transaction code, followed by zero or more member names, as required for transaction. There is at least one space between member names for those transactions, which involve more the one member

Example table for transactions(Facebook name as Phasebook)

Transaction	**Operational Description**
J Jack	Jack asks to join Phase Book
Q Ben	Ben quits Phase book
S Mann Jack	Mann sends a friend request to Jack
A Jack Mann	Jack accepts Mann's friend request
R Alice Ben	Alice reject Ben's friend request
U Bunny Seethal	Bunny un-friends Seethal
L Jack	List Jack's account, including his friends, his pending friend request sent and his pending requests received
L *	Special character '*' lists all user accounts

IV. PROPOSED IMPLEMENTATION

- **List Cell:** This defining a linked list node with two parts of information ***{name,*next}***
- **Name List:** This implements all operations of on names. All operations are specified as :

- Inserting any name to list of phase book

- Searching of existing members for a name

- Deleting any name of exisiting list

- Printing the list of names of members/friends/requests

- **Friends List:** This is list of friends
- **List of Invitations Received:** All friend requests list received from other members
- **List of Invitation Sent:** All friend requests sent to other members

V. PROPOSED ALGORITHM PROTOTYPES

i) List Cell Prototypes

- Initialize name and pointers
- Setter and getter functions

ii) Name List Prototypes

- Initialize names list
- insertName(<string parameter>)
- isFound(<string parameter>) returns true/false
- deleteName(<string parameter>)
- printList()

iii) Member prototypes

- Setter and Getter for Name
- setFriends(<NameList parameter>)
- setInvitationsReceived(<NameList parameter>)
- setInvitationsSent(<NameList parameter>)
- setNext(<member parameter>)
- getFriends() returns Names List
- getInvitationsReceived() returns Names List
- getInvitationsSent() returns Names List
- getNext() returns Member

VI. PROPOSED ALGORITHM DEFINITIONS

(List getters/setters algorithms)

Algorithm String getName():

1: Start
2: Return name
3: Stop

Algorithm String getNext():

1: Start
2: Return *next
3: Stop

Algorithm String setName(string name):

1: Start
2: Current->name:=name
3: Stop

*Algorithm String setNext(*next):*

1: Start
2: Current->next:=next
3: Stop

ii) Name List algorithms

Algorithm for insertName(string name):

1: Start
2: Allocate new *list
3: if current names==NULL Then
4: current names=new list
5: if isNameFound(name)=false then
6: Add the name to current list ending
[End if]
7: Stop

Algorithm for isFound(string name) returns true/false:

1: Start
2: Repeat loop until end of the name list

3: If (name is found) return true
[End Loop]
4: If not found after entire loop then
5: Return false
[End If]
6: Stop

Algorithm for deleteName(string name):

1: Start
2: Call function isFound(name)
3: If found is true then
4: Traverse to that name
5: Delete the name
6: Else
7: print "Error message"
[End if]
8: Stop

Algorithm for printList():

1: Start
2: Initialize T=names
3: Repeat loop until end of T
4: Print each T->getName()
5: T=T->getNext()
[End while]
6: Stop

(Members setter algorithms)

Algorithm for setName(string name):

1: Start
2: current->name:=name
3: Stop

*Algorithm for setFriends(namelist*friends):*

1: Start
2: current->friends:=friends
3: Stop

*Algorithm for setInvitationsRecvd(namelist *invit_recvd)*

1: Start
2: current->invidations_Recvd:= invit_recvd
3: Stop

*Algorithm for setInvitationsSent(namelist *invit_sent)*

1: Start
2: current->invidations_Sent:=invit_sent
3: Stop

(Members getter algorithms)

Algorithm string getName():

1: Start
2: Return current->name
3: Stop

Algorithm nameslist getFriends():

1: Start
2: Return current->friends

3: Stop

Algorithm namelist getInviationsRcvd()

1: Start
2: Return current->inviations_Recvd
3: Stop

Algorithm namelist getInviationsSent()

1: Start
2: Return current->inviations_Sent
3: Stop

Algorithm member getNext()

1: Start
2: Return current->next
3: Stop

(Members other algorithms)

*Algorithm for insertMemberList(member *m)*

1: Start
2: Allocate *newmember
3: call newmember->setName(m->getName())
4: call newmember->setFriends(m->getFriends())
5: call newmember->setInvitationsRcvd(m ->getInvitationsRcvd())
6: call newmember->setInvitationsSent(m->getInvitationsSent())
7: call newmember->setNext(nothing)
8: If member is the first member then
9: member=newmember
10:else
11: traverse upto end of members
12: add at end of members
[End If]
13: Stop

*Algorithm for printMembers(member *m):*

1: Start
2: Print member name
3: Print friends list by m->getFriends()
4: Print Friends invitation sent by m->getInvitationsSent()
5: Print Friends invitation received by m->getInvitationsRcvd()
6: Stop

Conclusion

This implementation can be done through any of the programming languages either by structural languages or object oriented languages.

Acknowledgment

The authors would like to thank the Raghu Engineering College for their infrastructural support and Andhra University, Dept of Computer Science and Systems Engineering to develop this work.

References

- "Data Structures and Program Design in C" by Kruse Robert L
- "Data Structures and Algorithm Analysis in C" by Mark Allen Weiss
- "Algorithms in C Parts 1-4″ by Robert Sedgewick
- https://en.wikibooks.org/wiki/Data_Structures/List_Structures