PROJECT DETAILS-

1.Introduction-

The main objective of this project is to built a social networking site in c++. A social networking site is an online platform that allows people to create a public account and interact with other people.

The social networking site concept not a new for all. We all aware and familiar with social networking concept in real world. Currently the facebook, instagram ,twitter and many other popular social networking site take place in each home.

In this social networking site project users can create free account and communicate with other users by making friends. A registered users can make their virtual world on and share good thoughts and ideas to other users by writing a post on this system.

After becoming a member, users can search people, make friends, send messages by all other facilities of our system.

2. Tools used-

Code Blocks

3. Dataset Description-

Data is used from some online website.

4. Features-

Searching : A registered user can search other user by name and send them friend request and view his profile and picture gallery and also send him messages.

Post : In this post facility user can share their thoughts to other user via writing a post and users can read and making reply by like post and also share the post in his account using share button.

Friend: A friend module is main base module for all social networking site. we all familiar with this word add friend in social networking world. The same functionality we have developed in our friend module, sending friend request to other user and request would be accepted or rejected by request received user.

Send Messages : All users can send and receive messages to and from his friends. All user have an inbox to keep the received messages and he can reply and delete message as per his need.

User Account : In Account section user can update his profile detail and can change login password.

5.Design of Project-

- Trees -

We have constructed a binary search tree by treating each person as a node. Also ,binary search will be used for login.

The tree will rebalance itself when we delete a user from the records.

We have used BST for user login because in order to search 1 billion user, we will just need 30 comparisons. Big O notation: O(n logn).

Indented tree: In this user can see visually how the BST looks.

Hashing –

We have used hashing to add and delete the user friends.

- Linked list (if collision, we add elements to the first place of the linked list and move the rest to the bottom) , when we make a hash table, we get a prime number as a table size (so it decreases collisions).

- If loading factor of hash table is more than 75 percent, then we rehash.

- In Rehashing, we double the size of the table, get the next prime number, set that as table size, copy all the elements to a temp array, rehash all the elements to the new hash table, and delete temp array.

In collisions, we count the number of objects in each collision and print the highest collision chain, Destructor for each table which deletes the element, frees the memory

Concept of class -

a).Person Class-In this class person details are managed such as name, age, password, gender, city and college.

b).User class-

We have 3 interfaces: user, user admin, admin.

User: In user menu a user after logging to his/her account can add new friends, can send message to a friend, can add his/her status update ,can his/her account settings which lead him to user admin menu.

User admin: Here, a User can change his profile settings like details of him/her such as age, city , college, e.t.c.

Admin: In this we can delete a person from our added records ,also we can print the hash table and can print the binary tree also.

Some concepts of file handling-

We have used file handling to read the file where person details are stored.

Vector-

We have used vector to add or delete user friends from their network.

6.Implementation Details:

1. Person Class-

Use of vector to store friends and set status of User.

```
class person
{
private:
    string name;
    string password;
    string gender;
    int age;
    string city;
    string college;
    vector<person*> friends;
    vector< string> status;
    map<string, vector<string> > text;
    bool deactivate;
```

Boolean function to Deactivate user ID.

```
public:
    person();
    person ( string name, string password, string gender, int age,  string city,  string college);
    void  setName( string item);
    void  setPassword(string password);
    void  setGender(string gender);
    void  setAge(int age);
    void  setCity(string city);
    void  setCollege( string college);
    void  setStatus( string newStatus);
    void  deactivateAccount( bool deactivate);
    void  addFriend(person &friendName);
    void deleteFriend(person &friendName);
    void  receiveMessage(person &friendName ,string message);
    void  sendMessage(person &friendName,  string message);
    string getName() const;
    string getPassword() const;
```

Get and Set function to assign and return values respectively.

```
string getPwd() const;
string  getGender() const;
int  getAge() const;
string  getCity() const;
string  getCollege() const;
bool getStatus() const;
string getDate();
vector<person *>  getFriends() const;
void  printFriends() const;
void  printStatus() const;
void  printMessages() const;
void  printPerson() const;
```

//Operator Overloading

```
 bool operator < (const person &otherObject)
 {
     return name < otherObject.getName();
 }
 bool operator > (const person &otherObject)
 {
     return name > otherObject.getName();
 }
bool operator == (const person &otherObject) {
     return this->name == otherObject.getName();
 }
 bool operator != (const person &otherObject)
 {
     return name != otherObject.getName();
 }
friend ostream &operator<<( ostream &output,  const person &P )
 {
     output << setw(20) << P.getName()<< "  "
            << setw(10) << P.getGender()
            << "     "<< P.getAge() << "      "
            << setw(20) << P.getCity() << "       "  << P.getCollege();
     return output;
 }
};
```

//Ostream to handle output stream

Constructor to call objects of class.

```
person::person ( )
{
    name = " ";
    password = " ";
    gender = " ";
    age = 0;
    city = " ";
    college = " ";
    deactivate = false;
};
person::person ( string name, string password, string gender, int age,  string city,  string college)
{
    this->name = name;
    this->password = password;
    this->gender = gender;
    this->age = age;
    this->city = city;
    this->college = college;
    deactivate = false;
};
```

This pointer ⟶ used to refer current class instance values.

Also can be used to pass current object as a parameter .

```
void person::setName( string name)
{
    this->name = name;
}
void person::setPassword( string password)
{
    this->password = password;
}
void  person::setGender(string gender)
{
    this->gender = gender;
}
void  person::setAge(int age)
{
    this->age = age;
}
void  person::setCity(string city)
{
    this->city = city;
}
void  person::setCollege( string college)
{
    this->college = college;
}
void  person::setStatus(string newStatus) {
    string timeNow  = getDate()+newStatus;
    status.push_back(timeNow + "\n");
}
```

Use of vector to push the values and erase the values for message sending, receiving and deleting friends.

```
void  person::deactivateAccount( bool deactivate) {
    this->deactivate = deactivate;
}
 void  person::receiveMessage(person&friendName ,string message)
{
    friendName.text[this->name].push_back(message);
};
```

```
void  person::deleteFriend(person &friendName)
{
    if (&friendName == this)
    {
        cout << "You cannot delete yourself." <<  endl;
        return;
    }
    for (int i = 0; i < friends.size(); i++ ) {
        if (friends[i]->name == friendName.name) {
            friends.erase(friends.begin() + i);
            friendName.deleteFriend(*this);
            return  ;
        }
    }
    cout << name << " and " << friendName.name << " are no longer friends." <<  endl;
}
```

2.Binary search Tree-

```
class BinarySearchTree
{

    struct node
    {
    itemType *data;
    node* left;
    node* right;
    };
    node *root;
  public:
    BinarySearchTree() {root = NULL;}
    bool isEmpty() const {return root == NULL;}
    typename BinarySearchTree<itemType>::node* _insert(node*&, itemType &d);
    void _insert(itemType &d);
    void remove(itemType &data);
    void _remove(node*p ,itemType &data);
    int getBalance(node* N);
    void delete_BST(node*);
    void inOrder(node*);
    void print_inOrder();
    void printIndent();
    void indented(node* p, int indent );
    itemType* _search(node*    ,itemType const &d);
    itemType* search(itemType const &d);
    int Height(node*);
```

It is balanced (using AVL trees upon insertion), remove (balances tree again using same AVL functions), insertion is recursive.

```
    void rotateLeftOnce(node*&);
    void rotateLeftTwice(node*&);
    void rotateRightOnce(node*&);
    void rotateRightTwice(node*&);
};
```

3. Hashing-

In the Hash Table we have stored Person names and we have taken a variable named loading factor such that if it is greater than 0.75 table will be rehashed.

```
class hashing
{
private:
    int tableSize;
    int numberofentries;
    int largestbucket;
    int numberofcollisions;
   struct item
    {
        person *hashedPerson ;
        item* next;
        item() { next = NULL; hashedPerson = NULL; };
    };
    item **HashTable;
public:
    hashing() { tableSize = 0; }
    hashing(int size);
    int Hash(string key);
    void addObject(person &personObj);
    int NumberOfItemsInIndex(int index);
    void PrintTable();
    void removeObject(person &personObj);
    void getPrime(int &num);
   void deleteentry(){numberofentries--;}
    double getLoadingfactor(){return (double)numberofentries/(double)tableSize;}
    int getNumberofentries(){return numberofentries;}
    int getTablesize(){ return tableSize;}
    void reHash();
    int getNumberofcollisions(){return numberofcollisions;}
};
```

4. User class-

```
class user{
public:
    BinarySearchTree<person> tree;
    person personArray[100];
    int count;
    user()
    {
    count=0;
    }
    void readFile(string textname)
    {
        string  name;
        string password;
        string gender;
        string age;
        string city;
        string college;
        string wasteSpace;
        ifstream infile;
        infile.open("name.txt");
        while(!infile.eof())
        {
            getline(infile, name);
            getline(infile, password);
             getline(infile, gender);
            getline(infile, age);
            getline(infile, city);
            getline(infile, college);
            getline(infile, wasteSpace);
            int Age = atoi(age.c_str());


            person tempPerson(name, password, gender, Age, city, college);
            personArray[count] = tempPerson;
            tree._insert(personArray[count]);
            count++;


        }
    }
```

// Vector for bst to get person details.

//atoi to convert string to int

a).User Menu-

Login to user Profile and then user can add new friends ,send them a message,Put their Status Updated….

```
cout << endl<< "-------------"<< temp->getName() << " Profile --------------" << endl;;
cout << "Choose an option from below" << endl;
cout << "1. Add a friend" << endl
<< "2. Write a new Status Message" << endl
<< "3. Send a message" << endl
<< "4. Show your friends" << endl
<< "5. Show newsfeed" << endl
<< "6. Print your status messages" <<endl
<< "7. Print your private messages" << endl
<< "8. Account Settings" << endl
<< "9. Logout"  << endl << "[Choose Your Option] : " ;
```

b).User Admin Menu-

Basically , in this user can update their account settings such as changing password or their details or can deactivate their profile for some time.

```
cout << "-------------"<< personName.getName() << " Settings --------------" << endl;;
cout << "Choose an option from below" << endl;
cout << "1. Change Password" << endl
<< "2. Change City " << endl
<< "3. Change University" << endl
<< "4. Delete a Friend" << endl
<< "5. Deactivate Profile " << endl
<< "6. Back to your Profile" << endl
```

c).Admin Menu-

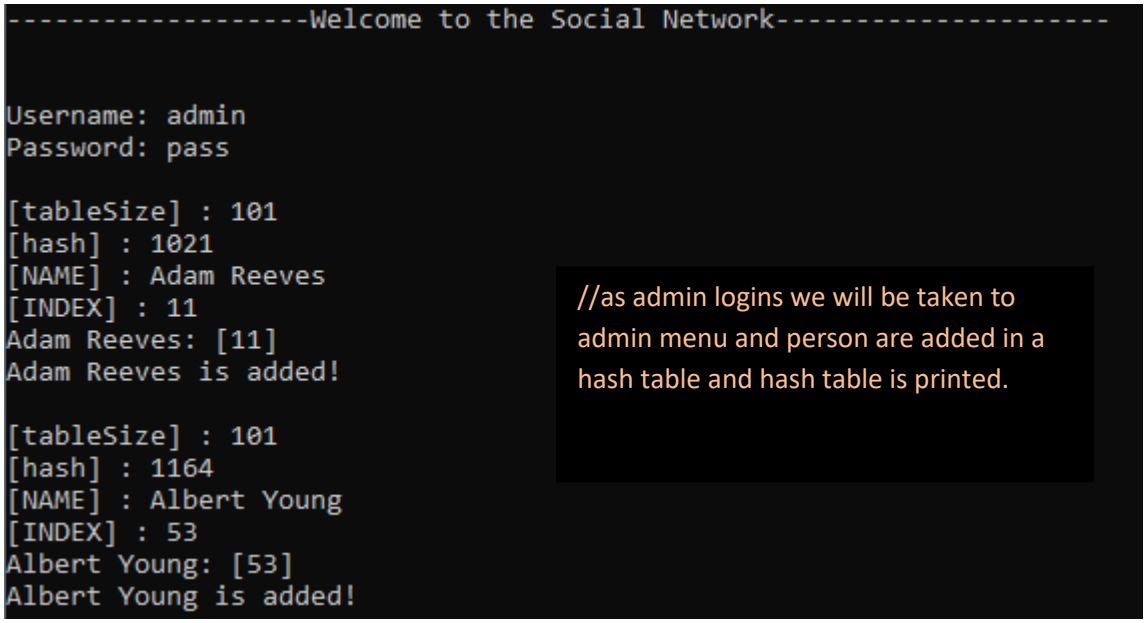
In this we can Delete a person from our records or we can print the binary tree and hash table for the added persons .

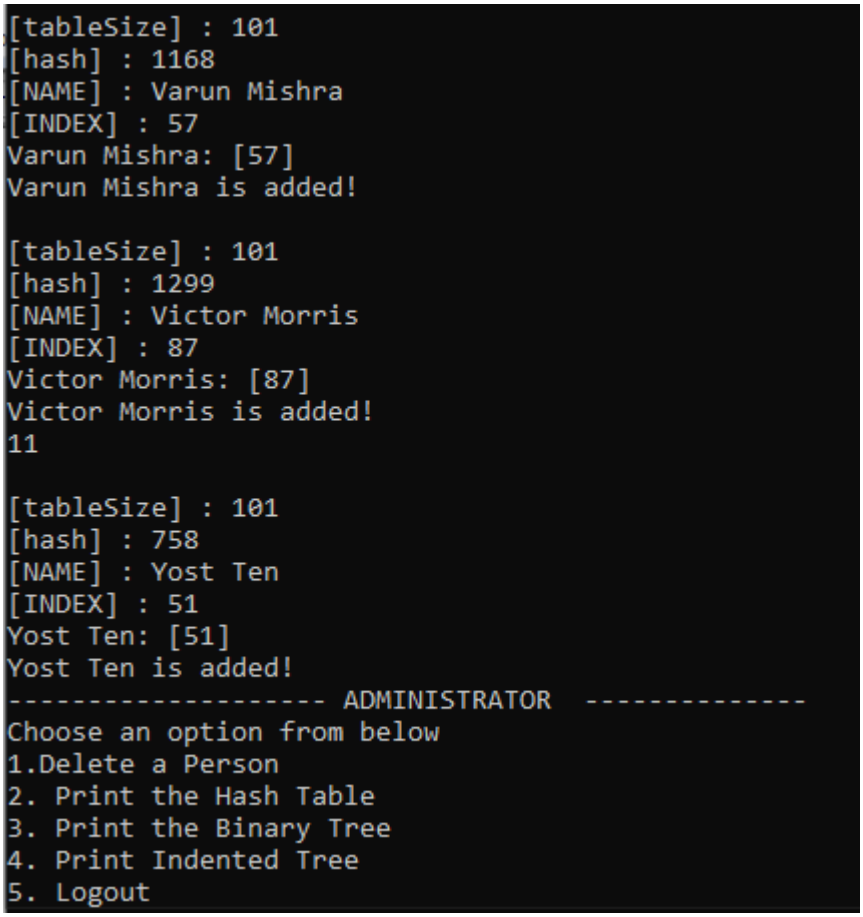
```
void adminMenu(user &obj, string password) {
    hashing hashfunc(count);
    for (int i = 0 ; i < count ; i++ ) {
        hashfunc.addObject(personArray[i]);
    }
    person *temp; person temporary; string name;


    if (password != "pass")
    {
        cout << "incorrect username or password";
        return;
    }

    bool options = true;
    while(options == true) {
        cout <<  "--------------------- ADMINISTRATOR  --------------" << endl;;
        cout << "Choose an option from below" << endl;
        cout << "1.Delete a Person" << endl
        << "2. Print the Hash Table" << endl
        << "3. Print the Binary Tree" << endl
        << "4. Print Indented Tree" <<endl
        << "5. Logout" << endl
        << "[Choose Your Option] : " ;
```

5.Main Menu-

```
int main()
{

    user u;
    u.readFile("name.txt");
    int input = 0;
    while(input == 0) {
        cout << "\n-------------------Welcome to the Social Network---------------------\n\n\n";
        string name;
        string password;
        cout << "Username: ";
        cin.ignore(1,'\n');                    // Use of File Handling to read user records file
        getline(cin, name);
        cout << "Password: ";
        cin>>password;
        if(name != "admin")
            u.userMenu(name, password);
        else
            u.adminMenu(u, password);
    }

}
```

Here , if user is admin then it will take us to admin menu and if user want to login to his/her profile and he/she will be taken to their profile.


------------------Welcome to the Social Network--------------------
Username: admin
Password: pass
[tableSize] : 101
[hash] : 1021
[NAME] : Adam Reeves
[INDEX] : 11
Adam Reeves: [11]
Adam Reeves is added!
[tableSize] : 101
[hash] : 1164
[NAME] : Albert Young
[INDEX] : 53
Albert Young: [53]
Albert Young is added!
//as admin logins we will be taken to admin menu and person are added in a hash table and hash table is printed.

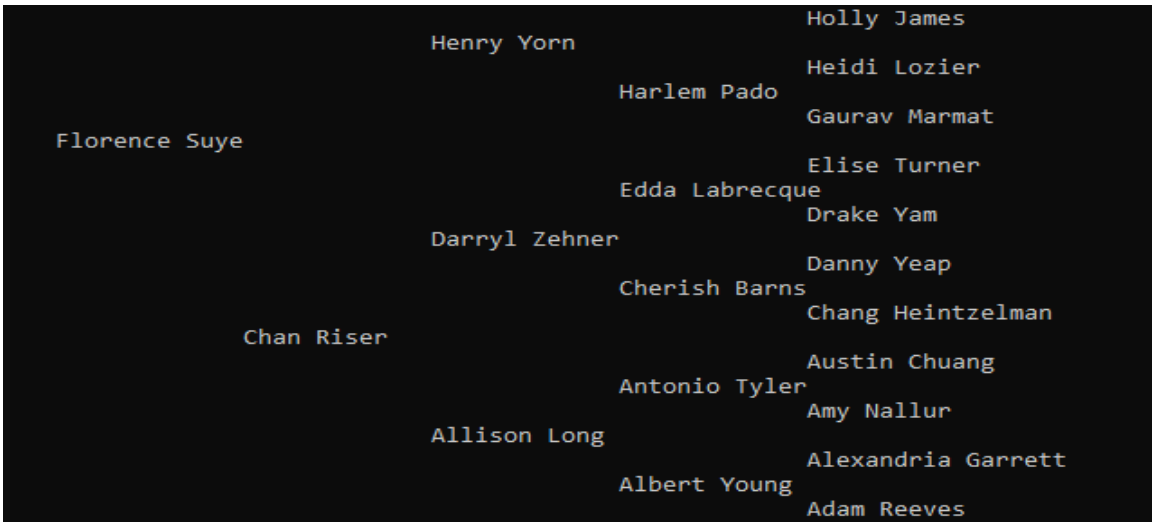

[tableSize] : 101
[hash] : 1168
[NAME] : Varun Mishra
[INDEX] : 57
Varun Mishra: [57]
Varun Mishra is added!
[tableSize] : 101
[hash] : 1299
[NAME] : Victor Morris
[INDEX] : 87
Victor Morris: [87]
Victor Morris is added!
11
[tableSize] : 101
[hash] : 758
[NAME] : Yost Ten
[INDEX] : 51
Yost Ten: [51]
Yost Ten is added!
-------------------- ADMINISTRATOR --------------
Choose an option from below
1.Delete a Person
2. Print the Hash Table
3. Print the Binary Tree
4. Print Indented Tree
5. Logout

Let us print the binary search tree created-

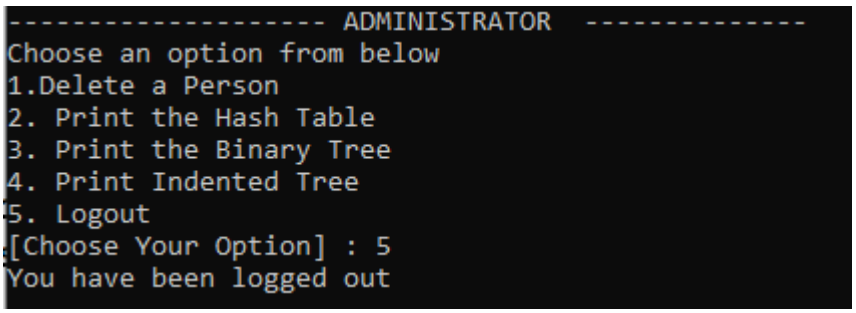
```
[Choose Your Option] : 3
****** PRINTING BINARY SEARCH TREE *******
         Luna Chong      Female     20          Ann Arbor, MI       University of Michigan
      Florence Suye      Female     20          Ann Arbor, MI       University of Michigan
         Chan Riser        Male     29        San Antonio, TX       University of Texas, San Antonio
       Allison Long      Female     23             Garland,TX       Amberton University
       Albert Young        Male     24              Dothan, AL      Wallace Community College
        Adam Reeves      Female     23             Phoenix, AZ      Ariona State University
  Alexandria Garrett     Female     14             Chicago, IL      N/A
      Antonio Tyler        Male     21             Belmont, TN      Belmont University, TN
         Amy Nallur      Female     22            Grinnell, IA      Grinnell College
      Austin Chuang        Male     22            Galveston, TX     University of Texas Medical Branch
      Darryl Zehner        Male     20            San Diego, CA     San Diego State University
      Cherish Barns        Male     34             Chicago, IL      Chiago Business School
   Chang Heintzelman     Female     17         Philadelphia, PA     Drexel Univerisity
         Danny Yeap        Male     21      Cupertino, California    De Anza College
     Edda Labrecque      Female     21              Asutin, TX      University of Texas, Austin
          Drake Yam        Male     24              Irvine, CA      Irvine Valley College
       Elise Turner      Female     22        New York, New York    NY State University
        Jerry Hayes        Male     21            Mishawaka, IN     Bethel College
         Henry Yorn        Male     20           Stony Brook, NY    Stony Brook University
        Harlem Pado        Male     24      College Station, TX     Texas A&M University
      Gaurav Marmat        Male     20             New York, NY     Texas A&M University
       Heidi Lozier      Female     23              Phoenix, AZ     Ariona State University
         Jake Mires        Male     21           Blacksburg, VA     Virginia Tech University
        Holly James      Female     20             New York, NY     Hunter College
           James Li        Male     26              Merced, CA      University of California Merced
          Joy Yen      Female     21             Killeen, TX       Central Texas College
        John Corner        Male     18          Arlington, Texas    Texas Instruments
```

```
        John Smith        Male     17         Philadelphia, PA     Drexel Univerisity
      Laura Jordan      Female     20            Ann Arbor, MI     University of Michigan
       Lam Ginger      Female     20           Stony Brook, NY    Stony Brook University
        Luis Warp        Male     24                 Davis, CA     University of California Davis
    Rebecca Ramirez     Female     20        Stonehill College     Easton, MA
       Molly Wharp      Female     22             Berkeley, CA     University of California Berkeley
    Manuel Peterson        Male     18            Palo Alto, CA     Stanford University
      Magaret Lisi      Female     20             New York, NY     University of Buffalo
               Mik          21      0          Beth Medrash .
    Phillip  Smith        Male     28       Berrien Springs, MI     Andrews University
       Noah Jones        Male     29           San Antonio, TX     University of Texas, San Antonio
      Rachel Smith      Female     17         Philadelphia, PA     Drexel Univerisity
          Tam Gern      Female     22           Stony Brook, NY    Stony Brook University
        Ron Gotor        Male     22          Scotts Valley, CA     Bethany University
      Richard James        Male     24                 Irvine, CA     Irvine Valley College
      Sumit Sharma        Male     21      Cupertino, California    De Anza College
         Tom Hanks        Male     20            San Diego, CA     San Diego State University
      Tiffani Sahr      Female     28            Los Angeles, CA     UCLA
      Victor Morris        Male     31             Fort Worth, TX     Texas State University
      Varun Mishra        Male     23      Sunnyvale, California    De Anza College
          Yost Ten        Male     22             Rochester, MN     Crossroads College
```

Let’s see how the Binary Search Tree Looks-



Let’s Logout and go to User Profile-



```
------------------Welcome to the Social Network--------------------


Username: Adam Reeves
Password: mada

-------------Adam Reeves Profile --------------
Choose an option from below
1. Add a friend
2. Write a new Status Message
3. Send a message
4. Show your friends
5. Show newsfeed
6. Print your status messages
7. Print your private messages
8. Account Settings
9. Logout
```

//logging in to Profile

```
[Choose Your Option] : 1
Which friend would you like to add
Yost Ten
Adam Reeves and Yost Ten are now friend

Press z to continue : z

-------------Adam Reeves Profile --------------
Choose an option from below
1. Add a friend
2. Write a new Status Message
3. Send a message
4. Show your friends
5. Show newsfeed
6. Print your status messages
7. Print your private messages
8. Account Settings
9. Logout
[Choose Your Option] : 1
Which friend would you like to add
Jake Mires
Adam Reeves and Jake Mires are now friend

Press z to continue : z
```

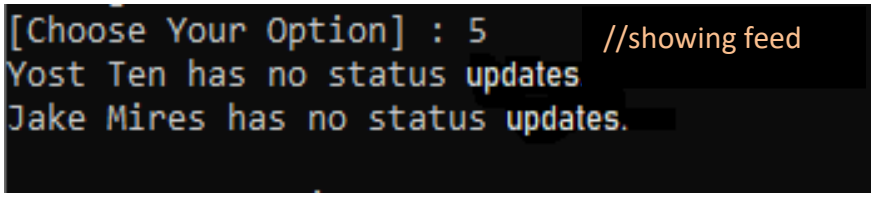
//adding Friends

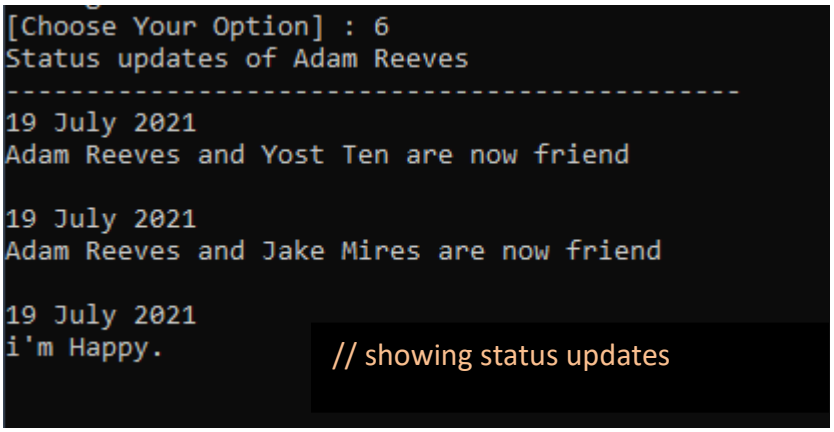
```
[Choose Your Option] : 2
What's on your mind?
i'm Happy.

Press z to continue : z
```

// updating status

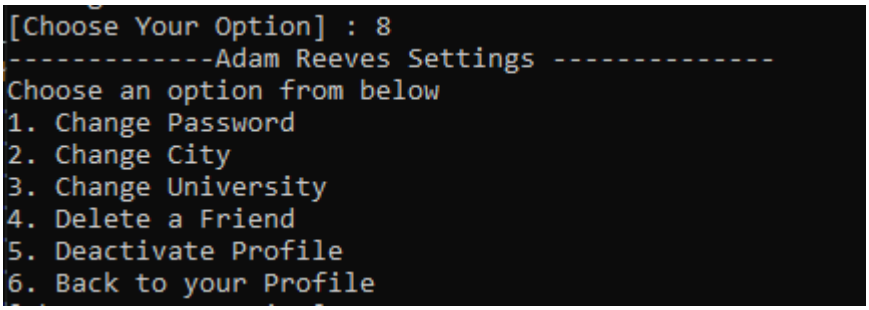
```
[Choose Your Option] : 3
Which friend would you like to send a message
Yost Ten
What message would you like to send :
Hii...Hope u are fine.

Press z to continue : z
```

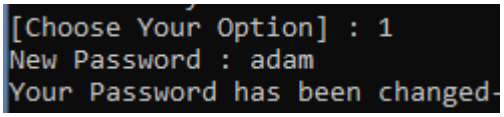
//sending message to friend



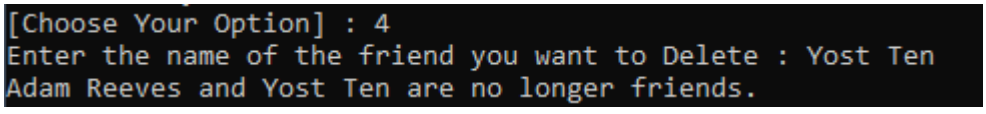


Changing account Settings……User Admin Menu



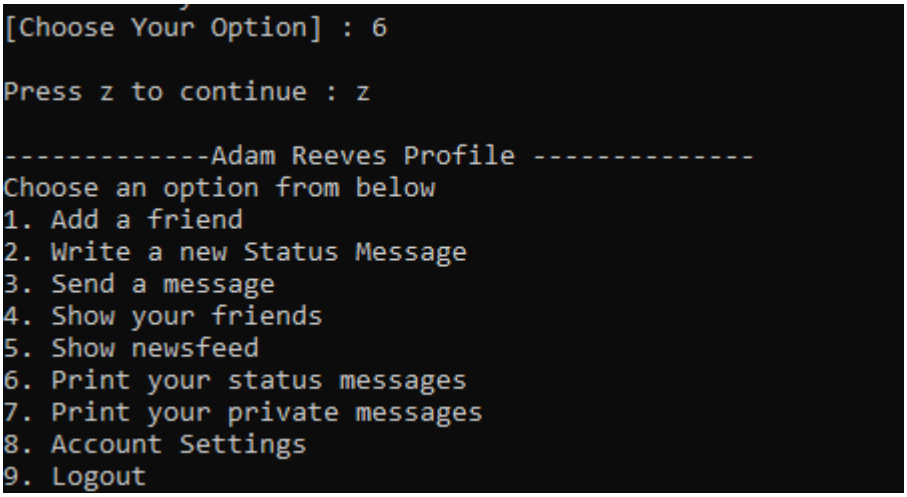


//Changing Password

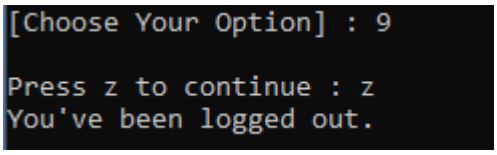


//Deleting Friend

Let’s Go back to Profile-



Let’s Logout-



Why we need a Self Balancing BST-

Most operations on a binary search tree (BST) take time directly proportional to the height of the tree, so it is desirable to keep the height small. A binary tree with height h can contain at most $2^0+2^1+\cdots+2^h = 2^{h+1}-1$ nodes. It follows that for any tree with n nodes and height h:

$$n \leq 2^{h+1} - 1$$

And that implies:

$$h \geq \lceil \log_2(n+1) - 1 \rceil \geq \lfloor \log_2 n \rfloor$$

In other words, the minimum height of a binary tree with n nodes is $\log_2(n)$, rounded down; that is, $\lfloor \log_2 n \rfloor$.

However, the simplest algorithms for BST item insertion may yield a tree with height n in rather common situations. For example, when the items are inserted in sorted key order, the tree degenerates into a linked list with n nodes. The

difference in performance between the two situations may be enormous: for example, when n = 1,000,000, the minimum height is $\lfloor \log_2(1,000,000) \rfloor = 19$.

If the data items are known ahead of time, the height can be kept small, in the average sense, by adding values in a random order, resulting in a random binary search tree. However, there are many situations (such as online algorithms) where this randomization is not viable.

Self-balancing binary trees solve this problem by performing transformations on the tree (such as tree rotations) at key insertion times, in order to keep the height proportional to $\log_2(n)$. Although a certain overhead is involved, it may be justified in the long run by ensuring fast execution of later operations.

While it is possible to maintain a BST with minimum height with expected O(log n) time operations (lookup/insertion/removal), the additional space requirements required to maintain such a structure tend to outweigh the decrease in search time. For comparison, an AVL tree is guaranteed to be within a factor of 1.44 of the optimal height while requiring only two additional bits of storage in a naive implementation. Therefore, most self-balanced BST algorithms keep the height within a constant factor of this lower bound.

In the asymptotic ("Big-O") sense, a self-balancing BST structure containing n items allows the lookup, insertion, and removal of an item in O(log n) worst-case time, and ordered enumeration of all items in O(n) time. For some implementations these are per-operation time bounds, while for others they are amortized bounds over a sequence of operations. These times are asymptotically optimal among all data structures that manipulate the key only through comparisons.