#include <stdio.h>

#include<conio.h>

Struct data{

int owner\_id;

int value;

string name;

};

struct Node {

Date timestamp;

Data data;

int nodeNumber;

String nodeId;

String referneceNodeId;

String genesisreferencenodeid;

Struct childnode \*node1,\*node2,\*node3;

String hashvalue;

String childreferencenodeid;

};

struct node \*head = NULL;

struct node \*current = NULL;

struct childnode{

Date timestamp;

Data data;

int nodeNumber;

String nodeId;

String referneceNodeId;

String genesisreferencenodeisd;

String parentId;

String hashvalue;

};

//addition of a genesis node key

void addnewnode(String dat,String time,String nodId ,int count,int ownerid,string name)

{

struct Node\* new\_node =(struct Node\*) malloc(sizeof(struct Node));

new\_node->data->value = dat;

new\_node->data->owner\_id = ownerid;

new\_node->data->name = name;

new\_node->timestamp = time;

new\_node->nodeId=nodId;

new\_node->nodeNumber=count;

new\_node->genesisreferencenodeid=null;

}

//add new node of child

void addnewchildnode(Struct node \*list,String dat,String time, String nodId,int count ,String parentnodeid,int ownerid,string name)

{

struct p\_node = find(list,parentnodeid)l;

if(p\_node->node1->data == 0 ||p\_node->node1->data != 0 || p\_node->node1->data != 0 )

struct childnode\* new\_node =(struct childnode\*) malloc(sizeof(struct Node));

new\_node->timestamp = time;

new\_node->nodeId=nodId;

new\_node->data->value = dat;

new\_node->data->owner\_id = ownerid;

new\_node->data->name = name;

new\_node->nodeNumber=count;

new\_node->referenceNodeId=parentnodeid;

new\_node->genesisreferencenodeid=null;

p\_node->node1 = childnode;

}

//find the parent node with specific node id

struct node \*find(struct node \*list, int id) {

if(list == NULL) {

return NULL;

} else {

return list->id == id ? list : find(list->next, id);

}

}

function uniqueid(){

// always start with a letter (for DOM friendlyness)

string idstr=String.fromCharCode(Math.floor((Math.random()\*25)+65));

do {

// between numbers and characters (48 is 0 and 90 is Z (42-48 = 90)

int ascicode=Math.floor((Math.random()\*42)+48);

if (ascicode<58 || ascicode>64){

// exclude all chars between : (58) and @ (64)

idstr+=String.fromCharCode(ascicode);

}

} while (idstr.length<32);

return (idstr);

}

//fetching the node id with the data value

struct node \*findwithdata(struct node \*list, int data12) {

if(list == NULL) {

return NULL;

} else {

return list->data->value == data12 ? list : findwithdata(list->next, data12);

}

}

//update data of any of the node includes two values old and new one

void update\_data(int old, int newq) {

int pos = 0;

if(head==NULL) {

printf("Linked List not initialized");

return;

}

current = head;

while(current->next!=NULL) {

if(current->data->value == old) {

current->data->value = newq;

printf("\n%d found at position %d, replaced with %d\n", old, pos, newq);

return;

}

current = current->next;

pos++;

}

printf("%d does not exist in the list\n", old);

}

string encryption(string message,int key)

{

char message[100], ch;

or(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch + key;

if(ch > 'z'){

ch = ch - 'z' + 'a' - 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch + key;

if(ch > 'Z'){

ch = ch - 'Z' + 'A' - 1;

}

message[i] = ch;

}

}

return message[];

}

string decryption(string message,int key)

{

char message[100], ch;

for(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch - key;

if(ch < 'a'){

ch = ch + 'z' - 'a' + 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch - key;

if(ch < 'A'){

ch = ch + 'Z' - 'A' + 1;

}

message[i] = ch;

}

}

return message;

}

void printList(struct Node \*head )

{

if (!head)

return;

while (head->next != NULL)

{

cout << head->data->value << " -> ";

head = head->next;

}

cout << head->data << endl;

}

// longest chain of any genesis node as well as any of the node

void longestConsecutiveUtillengthofanynode(Node\* root, int curLength,

int expected, int& res)

{

if (root == NULL)

return;

// if root data has one more than its parent

// then increase current length

if (root->data->value == expected)

curLength++;

else

curLength = 1;

// update the maximum by current length

res = max(res, curLength);

// recursively call left and right subtree with

// expected value 1 more than root data

longestConsecutiveUtillengthofanynode(root->left, curLength,

root->data + 1, res);

longestConsecutiveUtillengthofanynode(root->right, curLength,

root->data + 1, res);

}

// method returns length of longest consecutive

// sequence rooted at node root

int longestConsecutiveanynode(Node\* root)

{

if (root->data->value == 0)

return 0;

int res = 0;

// call utility method with current length 0

longestConsecutiveUtillengthofanynode(root, 0, root->data, res);

return res;

}

//merging the code of he two nodes

void merge(struct Node \*p, struct Node \*\*q)

{

struct Node \*p\_curr = p, \*q\_curr = \*q;

struct Node \*p\_next, \*q\_next;

// While therre are avialable positions in p checking if they are not null

while (p\_curr != NULL && q\_curr != NULL)

{

p->data->value = p->data->value +q->data->value;

deleteNode(\*q)

}

\*q = q\_curr; // Update head pointer of second list

}

void deleteNode( Node \* node )

{

Node \* temp = node->next;

node->data = node->next->data;

node->next = temp->next;

free(temp);

}

int main()

{

static int count =1;

string nodeid ;

struct node \*list;

string strq;

string data;

string data12;

string data2;

string key;

string dataqw;

struct node \*head;

string nodeidd;

string ans;

// string decrypmessg = "";

string encrmessg ="ebdh";

string mesg;

printf("user encryption and decryption");

printf("enter the mesage")

scanf("%c",&messg);

// printf("enter the decrypted message");

//scanf("%c",&decrypmessg);

printf("enter the key");

scanf("%d",&key2);

printf("while we are checking if you are the user are not ");

string enc = encryption(messg,key2);

if(enc == encrmesg)

{

printf("enter the key for dataencryption and decryption");

scanf("%d",&key);

printf("want to add a new genesis node")

scanf("%c",&strq);

//insert the genesis node id

if(strq == "yes")

{

printf("enter the data of the genesis node")

scanf("%c",&data);

string da = encryption(data,key);

nodeidd = uniqueid();

struct node \*head = nodeidd;

addnewnode(list,da,currenttime,nodeidd,count);

count++;

}

else {

printf("enter the data of the child node ")

scanf("%c",&data12);

printf("enter the data of the parentnode")

scanf("%c",&data2);

// struct node \*pnode = findwithdata(list,data2);

//for finding with id

//take input id from the user

//printf("enter the parent id ");

//scanf("%c",&parentid);

//struct node \*pnode = find(list,id);

struct node \*pnode = find(list,parentid);

if(p\_node->data->value < p\_node->node1->data->value + p\_node->node2->data->value +p\_node->node3->data->value +data12)

{

printf("cant be added as length of the node is ");

}

else {

int da = Integer.parseInt(encryption(data,key));

encryption(da,key);

addnewchildnode(list,data,currentime(),uniqueid(),count,p\_node->id)

count++;

}

}

else {

printf("you are not a valid user");

}

//list printing

printf("want to print the list")

scanf("%c",%ans);

if(ans=="yes")

{

printlist(nodeidd);

}

//update the value of the list

printf("want to update data");

printf("enter the old one and the new one");

scanf("%d %d ",&old,&new);

update\_data(old,new);

//find the longest chain of any of the node

printf("enter the value for the node which you want to find the maximum length ");

sacnf("%d",&dataqw);

struct childnode \*node = findwithdata(list,dataqw);

longestConsecutiveanynode(node);

//findi the longest node for any of the genesis node

printf("enter the value for the genesis node which you want to find the maximum length ");

sacnf("%d",&dataqw);

struct Node \*nodegen = findwithdata(list,dataqw);

longestConsecutiveanynode(nodegen);

//want to merge the two data fields

printf("enter the two data of the nodes to be merged");

scanf("%d %d ",&data1,&data24);

#include <stdio.h>

struct Node {

Date timestamp;

Struct data{

int owner\_id;

int value;

string name;

};

int nodeNumber;

String nodeId;

String referneceNodeId;

String genesisreferencenodeid;

Struct childnode \*node1,\*node2,\*node3;

String hashvalue;

String childreferencenodeid;

};

struct node \*head = NULL;

struct node \*current = NULL;

struct childnode{

Date timestamp;

Struct data{

int owner\_id;

int value;

string name;

};

int nodeNumber;

String nodeId;

String referneceNodeId;

String genesisreferencenodeisd;

String parentId;

String hashvalue;

};

//addition of a genesis node key

void addnewnode(String dat,String time,String nodId ,int count)

{

struct Node\* new\_node =(struct Node\*) malloc(sizeof(struct Node));

new\_node->data = dat;

new\_node->timestamp = time;

new\_node->nodeId=nodId;

new\_node->nodeNumber=count;

new\_node->genesisreferencenodeid=null;

}

//add new node of child

void addnewchildnode(Struct node \*list,String dat,String time, String nodId,int count ,String parentnodeid)

{

struct p\_node = find(list,parentnodeid)l;

if(p\_node->node1->data == 0 ||p\_node->node1->data != 0 || p\_node->node1->data != 0 )

struct childnode\* new\_node =(struct childnode\*) malloc(sizeof(struct Node));

new\_node->data = dat;

new\_node->timestamp = time;

new\_node->nodeId=nodId;

new\_node->nodeNumber=count;

new\_node->referenceNodeId=parentnodeid;

new\_node->genesisreferencenodeid=null;

p\_node->node1 = childnode;

}

//find the parent node with specific node id

struct node \*find(struct node \*list, int id) {

if(list == NULL) {

return NULL;

} else {

return list->id == id ? list : find(list->next, id);

}

}

function uniqueid(){

// always start with a letter (for DOM friendlyness)

string idstr=String.fromCharCode(Math.floor((Math.random()\*25)+65));

do {

// between numbers and characters (48 is 0 and 90 is Z (42-48 = 90)

int ascicode=Math.floor((Math.random()\*42)+48);

if (ascicode<58 || ascicode>64){

// exclude all chars between : (58) and @ (64)

idstr+=String.fromCharCode(ascicode);

}

} while (idstr.length<32);

return (idstr);

}

//fetching the node id with the data value

struct node \*findwithdata(struct node \*list, int data) {

if(list == NULL) {

return NULL;

} else {

return list->data == data ? list : find(list->next, data);

}

}

//update data of any of the node includes two values old and new one

void update\_data(int old, int new) {

int pos = 0;

if(head==NULL) {

printf("Linked List not initialized");

return;

}

current = head;

while(current->next!=NULL) {

if(current->data == old) {

current->data = new;

printf("\n%d found at position %d, replaced with %d\n", old, pos, new);

return;

}

current = current->next;

pos++;

}

printf("%d does not exist in the list\n", old);

}

string encryption(string message,int key)

{

char message[100], ch;

or(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch + key;

if(ch > 'z'){

ch = ch - 'z' + 'a' - 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch + key;

if(ch > 'Z'){

ch = ch - 'Z' + 'A' - 1;

}

message[i] = ch;

}

}

return message[];

}

string decryption(string message,int key)

{

char message[100], ch;

for(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch - key;

if(ch < 'a'){

ch = ch + 'z' - 'a' + 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch - key;

if(ch < 'A'){

ch = ch + 'Z' - 'A' + 1;

}

message[i] = ch;

}

}

return message;

}

string encryptionuser(string message,int key)

{

char message[100], ch;

or(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch + key;

if(ch > 'z'){

ch = ch - 'z' + 'a' - 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch + key;

if(ch > 'Z'){

ch = ch - 'Z' + 'A' - 1;

}

message[i] = ch;

}

}

return message[];

}

void printList(struct Node \*head )

{

if (!head)

return;

while (head->next != NULL)

{

cout << head->data << " -> ";

head = head->next;

}

cout << head->data << endl;

}

// longest chain of any genesis node as well as any of the node

void longestConsecutiveUtillengthofanynode(Node\* root, int curLength,

int expected, int& res)

{

if (root == NULL)

return;

// if root data has one more than its parent

// then increase current length

if (root->data == expected)

curLength++;

else

curLength = 1;

// update the maximum by current length

res = max(res, curLength);

// recursively call left and right subtree with

// expected value 1 more than root data

longestConsecutiveUtillengthofanynode(root->left, curLength,

root->data + 1, res);

longestConsecutiveUtillengthofanynode(root->right, curLength,

root->data + 1, res);

}

// method returns length of longest consecutive

// sequence rooted at node root

int longestConsecutiveanynode(Node\* root)

{

if (root->data == 0)

return 0;

int res = 0;

// call utility method with current length 0

longestConsecutiveUtillengthofanynode(root, 0, root->data, res);

return res;

}

//merging the code of he two nodes

void merge(struct Node \*p, struct Node \*\*q)

{

struct Node \*p\_curr = p, \*q\_curr = \*q;

struct Node \*p\_next, \*q\_next;

// While therre are avialable positions in p checking if they are not null

while (p\_curr != NULL && q\_curr != NULL)

{

p->data = p->data +q->data;

deleteNode(\*q)

}

\*q = q\_curr; // Update head pointer of second list

}

void deleteNode( Node \* node )

{

Node \* temp = node->next;

node->data = node->next->data;

node->next = temp->next;

free(temp);

}

void changeowner(string name ,int ownerid, Node \*root,int data)

{

if(root->data->value == data)

{

root->data->ownerid= ownerid;

root->data->name = name;

}

}

int main()

{

static int count =1;

string nodeid ;

struct node \*list;

string strq;

string data;

string data12;

string data2;

string key;

string dataqw;

struct node \*head;

string nodeidd;

string ans;

// string decrypmessg = "";

string encrmessg ="ebdh";

string mesg;

int ownerid;

string name;

printf("enter the user owner id ");

scanf("%d",&ownerid);

printf("enter the user name");

scanf("%c",&name);

printf("user encryption and decryption");

printf("enter the mesage")

scanf("%c",&messg);

// printf("enter the decrypted message");

//scanf("%c",&decrypmessg);

printf("enter the key");

scanf("%d",&key2);

printf("while we are checking if you are the user are not ");

string enc = encryptionuser(messg,key2);

if(enc == encrmesg)

{

printf("enter the key for dataencryption and decryption");

scanf("%d",&key);

printf("want to add a new genesis node")

scanf("%c",&strq);

//insert the genesis node id

if(strq == "yes")

{

printf("enter the data of the genesis node")

scanf("%c",&data);

string da = encryption(data,key);

nodeidd = uniqueid();

struct node \*head = nodeidd;

addnewnode(list,da,currenttime,nodeidd,count,ownerid,name);

count++;

}

else {

printf("enter the data of the child node ")

scanf("%c",&data12);

printf("enter the data of the parentnode")

scanf("%c",&data2);

// struct node \*pnode = findwithdata(list,data2);

//for finding with id

//take input id from the user

//printf("enter the parent id ");

//scanf("%c",&parentid);

//struct node \*pnode = find(list,id);

struct node \*pnode = find(list,parentid);

if(p\_node->data < p\_node->node1->data + p\_node->node2->data +p\_node->node3->data +data12)

{

printf("cant be added as length of the node is ");

}

else {

int da = Integer.parseInt(encryption(data,key));

encryption(da,key);

addnewchildnode(list,data,currentime(),uniqueid(),count,p\_node->id,ownerid,name)

count++;

}

}

else {

printf("you are not a valid user");

}

//list printing

printf("want to print the list")

scanf("%c",%ans);

if(ans=="yes")

{

printlist(nodeidd);

}

//update the value of the list

printf("want to update data");

printf("enter the old one and the new one");

scanf("%d %d ",&old,&new);

update\_data(old,new);

//find the longest chain of any of the node

printf("enter the value for the node which you want to find the maximum length ");

sacnf("%d",&dataqw);

struct childnode \*node = findwithdata(list,dataqw);

longestConsecutiveanynode(node);

//findi the longest node for any of the genesis node

printf("enter the value for the genesis node which you want to find the maximum length ");

sacnf("%d",&dataqw);

struct Node \*nodegen = findwithdata(list,dataqw);

longestConsecutiveanynode(nodegen);

//want to merge the two data fields

printf("enter the two data of the nodes to be merged");

scanf("%d %d ",&data1,&data24);

struct childnode \*node = findwithdata(list,dataq1);

struct childnode \*node2 = findwithdata(list,data24);

merge(\*node,\*node2);

//changing the owner of the node

printf("Change the owner of any of the node")

printf("enter the name of the owner ");

scanf("%c",&name);

printf("enter the id of the owner id ");

scanf("%d",&ownerid);

printf("enter the value of the node");

scanf("%d",&data);

//finding the node using the data and changing it

struct Node \*nodegen = findwithdata(list,data);

changeowner(name,ownerid,\*nodegen);

}