#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct NodeType;

typedef NodeType \* NodePtr;

struct recordType

{

long id;

string firstname;

string lastname;

double amount;

};

struct NodeType

{

long id;

string firstname;

string lastname;

double amount;

NodePtr frontLink;

NodePtr backLink;

};

class AccountList

{

private:

NodePtr head;

NodePtr node;

public:

AccountList();

void addAccountSorted(recordType record);

void updateAccount(recordType record);

void display(ofstream &lfstream);

};

//constructor for the accountlist (initializes values)

AccountList::AccountList()

{

head = new NodeType;

head->id = -1;

head->firstname = "";

head->lastname = "";

head->amount= -999.999;

head->frontLink=head;

head->backLink=head;

node = head;

}

//just a good ol' sorty mcsortyfunction

void AccountList::addAccountSorted(recordType record)

{

//Create the new node and fill it in.

NodePtr newPtr = new NodeType;

newPtr->id = record.id;

newPtr->firstname = record.firstname;

newPtr->lastname = record.lastname;

newPtr->amount = record.amount;

newPtr->frontLink = nullptr;

newPtr->backLink = nullptr;

//find the Node of point of insertion

NodePtr curr, prev;

for (curr=head->frontLink; curr!=head; curr=curr->frontLink)

{

if (record.id < curr->id) break;

}

//set prev

prev = curr->backLink;

//update the two forward links

newPtr->frontLink=prev->frontLink;

prev->frontLink = newPtr;

//update the two backward links

newPtr->backLink = curr->backLink;

curr->backLink = newPtr;

}

void AccountList::updateAccount(recordType record)

{

//move the node forward if at dummy node

if (node == head) node = node->frontLink;

//node is at the target node. do not move it

if (node->id == record.id)

{

//update the account

node->firstname = record.firstname;

node->lastname = record.lastname;

if(record.amount > 0)

{

node->amount += record.amount;

}

else

{

node->amount -= record.amount;

}

//if the account became zero or negative

//delete the node and move the node forward

if(node->amount <=0)

{

NodePtr temp = node;

node->backLink->frontLink = node->frontLink;

node->frontLink->backLink = node->backLink;

node = node->frontLink;

delete(temp);

}

}

else if (node->id <record.id)

{

while (node != head)

{

if (node->id >= record.id) break;

node = node->frontLink;

}

if (node->id == record.id)

{

//update the account

node->firstname = record.firstname;

node->lastname = record.lastname;

if(record.amount > 0)

{

node->amount += record.amount;

}

else

{

node->amount += record.amount;

}

//if the account became zero or negative

//delete the node and move the node forward

if(node->amount <=0)

{

NodePtr temp = node;

node->backLink->frontLink = node->frontLink;

node->frontLink->backLink = node->backLink;

node = node->frontLink;

delete(temp);

}

}

else

{

//insert the node prior to where node is.

NodePtr newPtr = new NodeType;

newPtr->id = record.id;

newPtr->firstname = record.firstname;

newPtr->lastname = record.lastname;

newPtr->amount = record.amount;

newPtr->frontLink = nullptr;

newPtr->backLink = nullptr;

newPtr->backLink = node->backLink;

newPtr->frontLink = node;

node->backLink->frontLink = newPtr;

node->frontLink->backLink = newPtr;

}

}

else

{

while (node != head )

{

if (node->id <= record.id) break;

node = node->backLink;

}

if (node->id == record.id)

{

//update the account

node->firstname = record.firstname;

node->lastname = record.lastname;

if(record.amount > 0)

{

node->amount += record.amount;

}

else

{

node->amount += record.amount;

}

//if the account became zero or negative

//delete the node and move the node forward

if(node->amount <=0)

{

NodePtr temp = node;

node->backLink->frontLink = node->frontLink;

node->frontLink->backLink = node->backLink;

node = node->frontLink;

delete(temp);

}

}

else

{

NodePtr newPtr = new NodeType;

newPtr->id = record.id;

newPtr->firstname = record.firstname;

newPtr->lastname = record.lastname;

newPtr->amount = record.amount;

newPtr->frontLink = nullptr;

newPtr->backLink = nullptr;

//first move the node forward by one

node = node->frontLink;

//This will make it point to the point of insertion node.

//Then insert the node prior to where node is.

newPtr->backLink = node->backLink;

newPtr->frontLink = node;

node->backLink->frontLink = newPtr;

node->frontLink->backLink = newPtr;

}

}

}

//This methodrecordeives an ofstream opened for the log file

//as a reference parameter and uses it to write the contents

//of the doubly linked list to the log file.

//This method can be used while performing updates.

void AccountList::display(ofstream & lfout)

{

for(NodePtr curr = head->frontLink; curr!=head; curr=curr->frontLink)

{

lfout << curr->id << " " << curr->firstname << " " << " " << curr->lastname << " " << curr->amount << endl;

}

}

int main()

{

recordType recordType;

AccountList accounts;

ofstream lfout ("log.txt");

string master\_file,transaction\_file;

cout<<" Enter the name of the master file :";

cin>>master\_file;

cout<<" Enter the name of the transaction file :";

cin>>transaction\_file;

ifstream master\_fin(master\_file.c\_str());

ifstream transaction\_fin(transaction\_file.c\_str());

if(master\_fin.is\_open())

{

while(!master\_fin.eof())

{

master\_fin>>recordType.id;

master\_fin>>recordType.firstname;

master\_fin>>recordType.lastname;

master\_fin>>recordType.amount;

accounts.addAccountSorted(recordType);

}

accounts.display(lfout);

if(transaction\_fin.is\_open())

{

while(!transaction\_fin.eof())

{

transaction\_fin>>recordType.id;

transaction\_fin>>recordType.firstname;

transaction\_fin>>recordType.lastname;

transaction\_fin>>recordType.amount;

accounts.updateAccount(recordType);

}

accounts.display(lfout);

}

else

{

cout<<" Unable to open "<<transaction\_file;

}

}

else cout<<" Unable to open "<<master\_file;

master\_fin.close();

transaction\_fin.close();

lfout.close();

}



