/\* -----------------------------------------------

Name: J-Zach Loke

Course: CMPS-385

Semester: Spring 2020

Project: No. 8 Part 1

Purpose: Given a text file, write a program to insert all data into an

ordered Linked List (based on account number).

----------------------------------------------- \*/

#include <iostream>

#include <string>

#include <fstream>

struct AccountNotFound : public std::exception

{

// custom exception

// thrown when a user tries to search for an account that does not exist.

const char \*msg () const throw ()

{

return "Could not find an account with that account number. Please Try again.";

}

};

struct node

{

// must be defined outside of the ordered linked list

// to avoid scope problems

int acc\_num;

std::string name;

float balance;

node \*prev;

node \*next;

};

class OrderedList

{

private:

node \*list;

public:

OrderedList() { list = NULL; }

void push(int acc, std::string n, float bal)

{

/\* name: push

input: int acc, string n, float bal

output: N/A

purpose: adds an element into the linked list stack and makes sure to keep it ordered. \*/

// create the node element we wish to insert

node \*insert = new(node);

insert->acc\_num = acc;

insert->name = n;

insert->balance = bal;

// temporary variables to help traversal

node \*curr = list; // represents the current node we're examining.

while (true)

{

// determines where in the list to insert the new node.

if (list == NULL)

{

// executes if the list is empty (ie. the first element is NULL)

list = insert;

insert->prev = NULL;

insert->next = NULL;

return;

}

else if (curr->acc\_num > insert->acc\_num)

{

// executes if the node we're examining has a value larger than the node we're inserting.

// if this executes, that means we've found a place within the list to insert our new node into.

if (curr->prev == NULL)

{

// if we're inserting to the front of the list

list = insert;

insert->prev = NULL;

}

else

{

insert->prev = curr->prev;

curr->prev->next = insert;

}

insert->next = curr;

curr->prev = insert;

return;

}

else if (curr->next == NULL)

{

// executes if we've reached the end of the list.

// if this executes, that means all nodes in the list have a smaller value than the node we're inserting.

// in this case, we should append the new node to the end of the list.

insert->next = NULL;

insert->prev = curr;

curr->next = insert;

return;

}

// executes when we haven't found the correct place to insert the new node.

curr = curr->next;

}

}

node\* pop()

{

/\* name: pop

input: N/A

output: node

purpose: removes and returns the smallest valued element from the linked list stack \*/

node \*p = list;

list = p->next;

return p;

}

bool isEmpty()

{

/\* name: isEmpty

input: N/A

output: bool

purpose: determines if the linked list stack is empty \*/

return (list == NULL) ? true : false;

}

void display()

{

/\* name: display

input: N/A

output: N/A

purpose: displays each account's account number and name \*/

node \*p = list;

while (p != NULL)

{

std::cout << '\t' << p->acc\_num << ',' << p->name;

p = p->next;

}

}

node\* getAccByNum(int n)

{

/\* name: getAccByNum

input: int n

output: node

purpose: finds and returns the account node whose account number matches the input

throws: AccountNotFound \*/

node \*p = list;

while (p != NULL)

{

if (p->acc\_num == n)

{

return p;

}

p = p->next;

}

throw AccountNotFound();

}

};

// function prototypes

OrderedList init();

void showBalance(OrderedList accounts);

void deposit(OrderedList accounts);

void withdraw(OrderedList accounts);

void showAccounts(OrderedList accounts);

void close(OrderedList accounts);

int main()

{

/\* name: main

input: N/A

output: N/A

purpose: main function to drive the program \*/

OrderedList accounts = init();

const std::string MENU = "---------------Menu-------------------\n1. Show my balance\n2. Deposit in my account\n3. Withdraw from my account\n4. Show me all\n5. Close my account\n\n";

std::cout << MENU;

void (\*funcs[]) (OrderedList) = {showBalance, deposit, withdraw, showAccounts, close};

char cont = 'y';

while (cont == 'y')

{

int choice;

std::cout << "Please enter your choice (1-5): ";

std::cin >> choice;

funcs[choice-1](accounts);

std::cout << std::endl << "CONTINUE(y/n)? ";

std::cin >> cont;

std::cout << std::endl;

}

system("pause");

return 0;

}

OrderedList init()

{

/\* name: init

input: N/A

output: OrderedList

purpose: inits the ordered list with data from input.txt \*/

OrderedList accounts = OrderedList();

// given input file

std::fstream file;

file.open("input.txt", std::ios::in);

while (!file.eof())

{

int acc;

std::string n;

float bal;

file >> acc >> n >> bal;

accounts.push(acc, n, bal);

}

return accounts;

}

void showBalance(OrderedList accounts)

{

/\* name: showBalance

input: OrderedList accounts

output: N/A

purpose: Shows an account's balance \*/

int num;

std::cout << "\tEnter your account number: ";

std::cin >> num;

try

{

node \*acc = accounts.getAccByNum(num);

std::cout << "\t\t" << acc->name << ", your balance is " << acc->balance << std::endl;

}

catch (AccountNotFound& e)

{

std::cout << e.msg() << std::endl;

}

}

void deposit(OrderedList accounts)

{

/\* name: deposit

input: OrderedList accounts

output: N/A

purpose: Deposits money into an account \*/

int num;

std::cout << "\tEnter your account number: ";

std::cin >> num;

try

{

node \*acc = accounts.getAccByNum(num);

float amount;

std::cout << "\t\t" << acc->name << ", how much would you like to deposit? ";

std::cin >> amount;

acc->balance += amount;

std::cout << "\t\t" << acc->name << ", your new balance is " << acc->balance << std::endl;

}

catch (AccountNotFound& e)

{

std::cout << e.msg() << std::endl;

}

}

void withdraw(OrderedList accounts)

{

/\* name: withdraw

input: OrderedList accounts

output: N/A

purpose: Withdraws money into an account \*/

int num;

std::cout << "\tEnter your account number: ";

std::cin >> num;

try

{

node \*acc = accounts.getAccByNum(num);

float amount;

std::cout << "\t\t" << acc->name << ", how much would you like to withdraw? ";

std::cin >> amount;

acc->balance -= amount;

std::cout << "\t\t" << acc->name << ", your new balance is " << acc->balance << std::endl;

}

catch (AccountNotFound& e)

{

std::cout << e.msg() << std::endl;

}

}

void showAccounts(OrderedList accounts)

{

/\* name: showAccounts

input: OrderedList accounts

output: N/A

purpose: Shows all accounts \*/

accounts.display();

std::cout << std::endl;

}

void close(OrderedList accounts)

{

/\* name: close

input: OrderedList accounts

output: N/A

purpose: Closes (deletes) an account \*/

int num;

std::cout << "\tEnter your account number: ";

std::cin >> num;

try

{

node \*acc = accounts.getAccByNum(num);

std::string name = acc->name;

acc->prev->next = acc->next;

delete acc;

std::cout << "\t\t" << name << ", we closed your account" << std::endl;

}

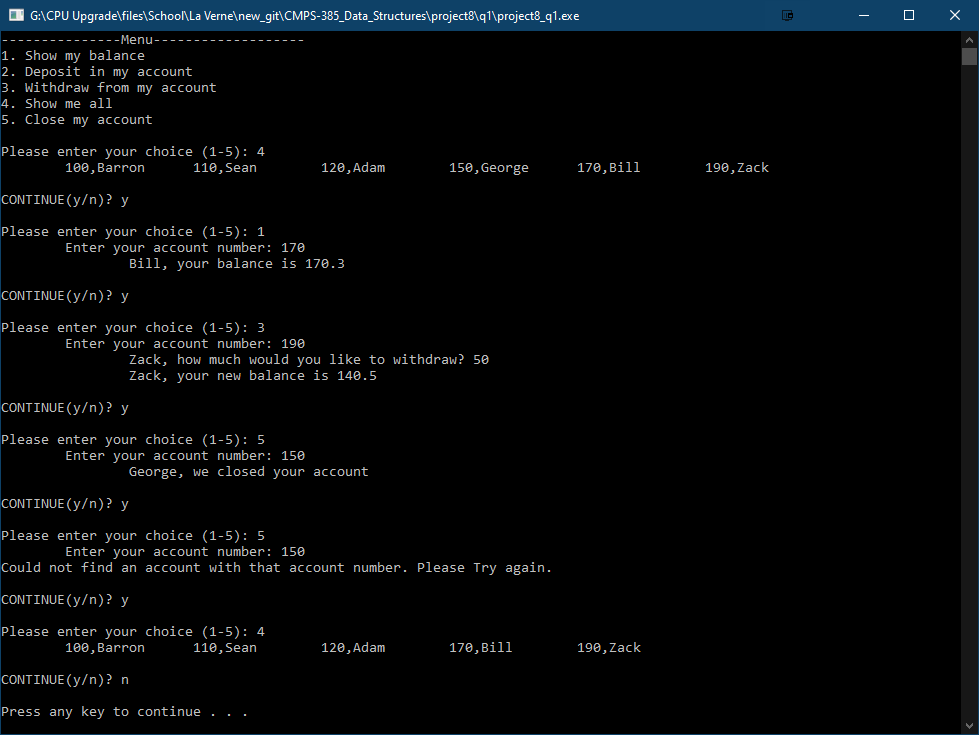
catch (AccountNotFound& e)

{

std::cout << e.msg() << std::endl;

}

}



1. Answer the following review questions

|  |  |
| --- | --- |
| Convert this expression to postfix  a\*( b – c\*(a+b) )  **bcab+\*-a\*** | Evaluate the following expression  + + 4 5 + 5 \* 2 3  **20** |
| What is the Big-of estimation of  (n+2)3\*(n-1)2 +(n+3)6 + log( a7+2n)  **O(n^6)** | What is the run-time of this function  Int f( int n )  { if (n==1) return 1;  return f(n-1)+f(n-2);  }  **O(2^n)** |
| Trace the following function  void f(int n)  { if (n>=2)  { cout<<n; f(n-1); cout<<n; f(n-2); }  }  **Given what calling statement?**  **I’ll assume f(4).**  **4 3 2 2 3 4 2** | Given int a[4]={9,5,8,2 }; what is printed  int \*p=a; int \*q=&a[3];  while(q>=p)  { cout<<\*q<<” “; --q; }    **2 8 5 9** |

1. Given the following linked lists and declarations struct node

List1 { int info;

node \*next;

};

List2 struct Node

{ int info;

Node \*next, \*back;

};

Write program segments to

|  |  |  |
| --- | --- | --- |
| Convert List1 to a circular linked list  node \*p = List1;  while (p->next != NULL)  {  p = p->next;  }  p->next = List1; | Insert a new node with info=10 at the rear of circular List1  node \*insert = new(node);  node->info = 10;  insert->next = List1;  while (p->next != List1)  { p = p->next; }  p->next = insert; | Count the number of nodes in circular List1  int count = 0;  node \*p = List1;  do  {  count++;  p = p->next;  } while (p != List1); |
| Use a for-loop to display the circular List1 four times  for (int i = 0; i < 4; ++i)  {  node \*p = List1;  do  {  cout << p->info << ‘\t’;  p = p->next;  } while (p != List1);  } | Count how many nodes are in List2  int count = 0;  node \*p = List2;  while (p != NULL)  {  count++;  p = p->next;  } | Display nodes in List2 from right-to-left  node \*p = List2;  while (p->next != NULL)  { p = p->next; }  while (p != NULL)  {  cout << p->info;  p = p->back;  } |
| In List2, go from left-to-right and stop at the first EVEN info. Go from right-to-left and stop at the first ODD info. Which number is larger?  int even = -1;  int odd = -1;  node \*p = List2;  if (p->info % 2 == 0)  {  even = p->info;  }  while (p->next != NULL)  {  p = p->next;  if (p->info % 2 == 0 &&  even != -1)  {  even = p->info;  }  }  while (p != NULL)  {  if (p->info % 2 == 1)  {  odd = p->info;  return;  }  }  cout << ((even > odd) ? even : odd); | This project is due on April 10 at 5:00 PM. Please email your program to Amber. I know thank you is not enough, but wanted to add my thanks for your cooperation. Ray | |