#include <iostream>

#include <string>

using namespace std;

class Expense {

public:

int expenseId;

string expenseDescription;

float expenseAmount;

Expense\* nextExpense;

Expense(int id, string description, float amount) {

this->expenseId = id;

this->expenseDescription = description;

this->expenseAmount = amount;

this->nextExpense = nullptr;

}

};

class ExpenseManager {

private:

Expense\* headExpense;

int nextExpenseId;

public:

ExpenseManager() {

headExpense = nullptr;

nextExpenseId = 1;

}

void addExpense(string description, float amount) {

Expense\* newExpense = new Expense(nextExpenseId++, description, amount);

if (headExpense == nullptr) {

headExpense = newExpense;

} else {

Expense\* current = headExpense;

while (current->nextExpense != nullptr)

current = current->nextExpense;

current->nextExpense = newExpense;

}

cout << "Expense added successfully!\n";

}

void viewAllExpenses() {

if (headExpense == nullptr) {

cout << "No expenses found.\n";

return;

}

Expense\* current = headExpense;

cout << "\nList of Expenses:\n";

while (current != nullptr) {

cout << "ID: " << current->expenseId

<< " | Description: " << current->expenseDescription

<< " | Amount: $" << current->expenseAmount << "\n";

current = current->nextExpense;

}

}

void updateExpense(int id, string newDescription, float newAmount) {

Expense\* current = headExpense;

while (current != nullptr) {

if (current->expenseId == id) {

current->expenseDescription = newDescription;

current->expenseAmount = newAmount;

cout << "Expense updated successfully!\n";

return;

}

current = current->nextExpense;

}

cout << "Expense with ID " << id << " not found.\n";

}

void deleteExpense(int id) {

if (headExpense == nullptr) {

cout << "No expenses to delete.\n";

return;

}

if (headExpense->expenseId == id) {

Expense\* toDelete = headExpense;

headExpense = headExpense->nextExpense;

delete toDelete;

cout << "Expense deleted successfully!\n";

return;

}

Expense\* current = headExpense;

while (current->nextExpense != nullptr && current->nextExpense->expenseId != id) {

current = current->nextExpense;

}

if (current->nextExpense == nullptr) {

cout << "Expense with ID " << id << " not found.\n";

} else {

Expense\* toDelete = current->nextExpense;

current->nextExpense = current->nextExpense->nextExpense;

delete toDelete;

cout << "Expense deleted successfully!\n";

}

}

void showHighestExpense() {

if (headExpense == nullptr) {

cout << "No expenses available to evaluate.\n";

return;

}

Expense\* current = headExpense;

Expense\* highestExpense = headExpense;

while (current != nullptr) {

if (current->expenseAmount > highestExpense->expenseAmount) {

highestExpense = current;

}

current = current->nextExpense;

}

cout << "\nHighest Expense:\n";

cout << "Description: " << highestExpense->expenseDescription

<< " | Amount: $" << highestExpense->expenseAmount << "\n";

}

};

int main() {

ExpenseManager expenseTracker;

int userChoice, expenseId;

string expenseDescription;

float expenseAmount;

do {

cout << "\n===== Expense Tracker Menu =====\n";

cout << "1. Add Expense\n";

cout << "2. View Expenses\n";

cout << "3. Update Expense\n";

cout << "4. Delete Expense\n";

cout << "5. Show Highest Expense\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> userChoice;

cin.ignore();

switch (userChoice) {

case 1:

cout << "Enter description: ";

getline(cin, expenseDescription);

cout << "Enter amount: ";

cin >> expenseAmount;

expenseTracker.addExpense(expenseDescription, expenseAmount);

break;

case 2:

expenseTracker.viewAllExpenses();

break;

case 3:

cout << "Enter expense ID to update: ";

cin >> expenseId;

cin.ignore();

cout << "Enter new description: ";

getline(cin, expenseDescription);

cout << "Enter new amount: ";

cin >> expenseAmount;

expenseTracker.updateExpense(expenseId, expenseDescription, expenseAmount);

break;

case 4:

cout << "Enter expense ID to delete: ";

cin >> expenseId;

expenseTracker.deleteExpense(expenseId);

break;

case 5:

expenseTracker.showHighestExpense();

break;

case 6:

cout << "Exiting Expense Tracker. Goodbye!\n";

break;

default:

cout << "Invalid choice. Please try again.\n";

}

} while (userChoice != 6);

return 0;

}