#include <iostream>

#include <string>

using namespace std;

// Class for a node in the linked list

class ExpenseNode {

public:

int id;

string description;

float amount;

ExpenseNode\* next;

ExpenseNode(int id, string desc, float amt) {

this->id = id;

this->description = desc;

this->amount = amt;

this->next = nullptr;

}

};

// Class for Expense Tracker

class ExpenseTracker {

private:

ExpenseNode\* head;

int nextId;

public:

ExpenseTracker() {

head = nullptr;

nextId = 1;

}

// Create

void addExpense(string description, float amount) {

ExpenseNode\* newNode = new ExpenseNode(nextId++, description, amount);

if (head == nullptr) {

head = newNode;

} else {

ExpenseNode\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

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

}

// Read

void viewExpenses() {

if (head == nullptr) {

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

return;

}

ExpenseNode\* temp = head;

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

while (temp != nullptr) {

cout << "ID: " << temp->id

<< " | Description: " << temp->description

<< " | Amount: $" << temp->amount << "\n";

temp = temp->next;

}

}

// Update

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

ExpenseNode\* temp = head;

while (temp != nullptr) {

if (temp->id == id) {

temp->description = newDescription;

temp->amount = newAmount;

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

return;

}

temp = temp->next;

}

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

}

// Delete

void deleteExpense(int id) {

if (head == nullptr) {

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

return;

}

if (head->id == id) {

ExpenseNode\* toDelete = head;

head = head->next;

delete toDelete;

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

return;

}

ExpenseNode\* temp = head;

while (temp->next != nullptr && temp->next->id != id) {

temp = temp->next;

}

if (temp->next == nullptr) {

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

} else {

ExpenseNode\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

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

}

}

// Show highest expense

void showHighestExpense() {

if (head == nullptr) {

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

return;

}

ExpenseNode\* temp = head;

ExpenseNode\* highest = head;

while (temp != nullptr) {

if (temp->amount > highest->amount) {

highest = temp;

}

temp = temp->next;

}

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

cout << "Description: " << highest->description << " | Amount: $" << highest->amount << "\n";

}

};

// Main function

int main() {

ExpenseTracker tracker;

int choice, id;

string description;

float amount;

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 >> choice;

cin.ignore();

switch (choice) {

case 1:

cout << "Enter description: ";

getline(cin, description);

cout << "Enter amount: ";

cin >> amount;

tracker.addExpense(description, amount);

break;

case 2:

tracker.viewExpenses();

break;

case 3:

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

cin >> id;

cin.ignore();

cout << "Enter new description: ";

getline(cin, description);

cout << "Enter new amount: ";

cin >> amount;

tracker.updateExpense(id, description, amount);

break;

case 4:

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

cin >> id;

tracker.deleteExpense(id);

break;

case 5:

tracker.showHighestExpense();

break;

case 6:

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

break;

default:

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

}

} while (choice != 6);

return 0;

}