This is a complete **Expense Tracking System** implemented in C++ using linked lists and variant data types. Here's the detailed explanation:

**1. Core Data Structures**

**Node Structure:**

cpp4 lines

* struct Node {
* variant<int, string> data; // Stores either integer or string
* Node\* next; // Pointer to next node
* };

Each node can hold either an **int** (for ID/Price) or **string** (for Item/Date)

**ExpenseList Class:**

cpp5 lines

* class ExpenseList {
* Node\* head; // Start of linked list
* Node\* tail; // End of linked list
* // ... methods ...
* };

Manages a linked list of exactly 4 nodes per expense

Implements RAII (Resource Acquisition Is Initialization) for memory safety

**2. Key Functionality**

**a. List Management**

* **Appending Nodes:** **append()** adds nodes in fixed order (ID → Name → Price → Date)
* **Deep Copying:** Copy constructor/assignment operator prevent shared nodes
* **Memory Management:** Destructor cleans up all nodes automatically

**b. Display System**

cpp4 lines

* void display() const {
* string labels[] = {"ID", "Item", "Price", "Date"};
* // ... prints all nodes with labels ...
* }

Shows all 4 fields of an expense

Uses **std::visit** for type-safe value access

**c. Price Calculation**

cpp4 lines

int getPrice() const {

// Navigates to 3rd node (price)

return price value or 0 if invalid

}

**3. User Interaction**

**Menu Options:**

1. **Add Expense:** Creates new 4-node linked list
2. **View All:** Displays all expenses
3. **Generate Report:** Shows total expenses and sum
4. **View Single:** Display specific expense
5. **Delete:** Remove expense by index
6. **Edit:** Modify individual fields
7. **Exit:** Clean up and quit

**Editing Mechanism:**

cpp4 lines

void updateNode(int position, const variant<int, string>& newValue) {

// Updates node at specified position (0-3)

// Uses try-catch for type safety

}

**4. Input Handling**

**Validation Functions:**

* **getIntInput()**: Ensures numeric input
* **getStringInput()**: Handles text with spaces
* **clearInputBuffer()**: Prevents input errors

**Example Validation:**

cpp2 lines

int id = getIntInput("Enter ID: "); // Rejects non-numbers

string name = getStringInput("Enter item name: "); // Allows spaces

**5. Memory Management**

**Key Features:**

1. **Deep Copying:**

cpp2 lines

ExpenseList(const ExpenseList& other) // Copy constructor

operator= // Assignment operator

1. **Automatic Cleanup:**

cpp1 lines

~ExpenseList() { /\* deletes all nodes \*/ }

1. **STL Vector Usage:**

cpp1 lines

vector<ExpenseList> expenses; // Manages expense lifetimes

**6. Error Handling**

* **Type Safety:** **try-catch** for variant access errors
* **Bounds Checking:** Validates menu choices and expense indices
* **Empty List Checks:** Handles operations on empty lists

**7. Typical Workflow**

1. **Initialization:**

cpp8 lines

EXPENSE TRACKING SYSTEM

Enter number of initial expenses to add: 2

Expense #1

Enter ID: 101

Enter item name: Groceries

Enter price: 85

Enter date (DD/MM/YYYY): 01/06/2024

1. **Editing:**

1-MENU: 6 (Edit)

2-Enter expense number to edit: 1

3-Select field to edit: 3

4-Enter new price: 95

1. **Reporting:**

1-TOTAL EXPENDITURE REPORT

2-Total expenses: 2

3-Total amount spent: $180

**8. Compilation & Execution**

**Requires C++17** (for variant/visit):

bash

1-g++ -std=c++17 expense\_tracker.cpp -o tracker

This implementation provides a robust, user-friendly expense tracking solution with proper memory management and input validation. ⬤