Full Code:

```
#include <iostream>

#include <vector>

#include <map>

#include <memory>

#include <fstream>

#include <iomanip>

#include <limits>

#include <stdexcept>

using namespace std;


// Enums for transaction types and categories

enum class TransactionType { INCOME, EXPENSE };

const vector<string> CategoryNames = {"Food", "Transport", "Utilities", "Medical", "Salary",
"Other"};


// Base Transaction Class

class Transaction {

protected:

    string date;

    string category;

    double amount;

    TransactionType type;
```

```
public:
    Transaction(const string& d, const string& c, double a, TransactionType t)
        : date(d), category(c), amount(a), type(t) {}

    virtual ~Transaction() = default;
    virtual void display() const = 0;

    string getCategory() const { return category; }
    double getAmount() const { return amount; }
    string getDate() const { return date; }
    TransactionType getType() const { return type; }
    bool isExpense() const { return type == TransactionType::EXPENSE; }
    bool isIncome() const { return type == TransactionType::INCOME; }
};

// Expense class
class Expense : public Transaction {
    string payee;
public:
    Expense(const string& d, const string& c, double a, const string& p)
        : Transaction(d, c, a, TransactionType::EXPENSE), payee(p) {}

    void display() const override {
        cout << "[Expense] " << date << " | " << setw(12) << left << category
             << " | $" << setw(8) << fixed << setprecision(2) << amount
             << " | Payee: " << payee << endl;
    }

    string getPayee() const { return payee; }
};

// Income class
class Income : public Transaction {
    string source;
public:
    Income(const string& d, const string& c, double a, const string& s)
        : Transaction(d, c, a, TransactionType::INCOME), source(s) {}

    void display() const override {
        cout << "[Income]  " << date << " | " << setw(12) << left << category
             << " | $" << setw(8) << fixed << setprecision(2) << amount
             << " | Source: " << source << endl;
    }

    string getSource() const { return source; }
};
```

```
// Budget class

class Budget {

    double totalLimit;

    map<string, double> categoryLimits;


public:

    Budget() : totalLimit(0.0) {}


    void setTotalLimit(double limit) {

        if (limit < 0) throw invalid_argument("Budget limit cannot be negative");

        totalLimit = limit;

    }


    void setCategoryLimit(const string& category, double limit) {

        if (limit < 0) throw invalid_argument("Category limit cannot be negative");

        categoryLimits[category] = limit;

    }


    double getCategoryLimit(const string& category) const {

        return categoryLimits.count(category) ? categoryLimits.at(category) : 0.0;

    }
    double getTotalLimit() const { return totalLimit; }

    bool isWithinCategoryLimit(const string& category, double spent) const {
        return !categoryLimits.count(category) || spent <= categoryLimits.at(category);
    }

    void display() const {
        cout << "\n--- Budget Limits ---\n";
        cout << "Total Budget: $" << fixed << setprecision(2) << totalLimit << "\n";
        cout << "Category Limits:\n";
        for (const auto& pair : categoryLimits) {
    cout << "  " << setw(12) << left << pair.first << ": $" << pair.second << "\n";
}

    }
};

// Tracker class
class Tracker {
    vector<unique_ptr<Transaction>> transactions;

public:
    void addTransaction(unique_ptr<Transaction> transaction) {
```

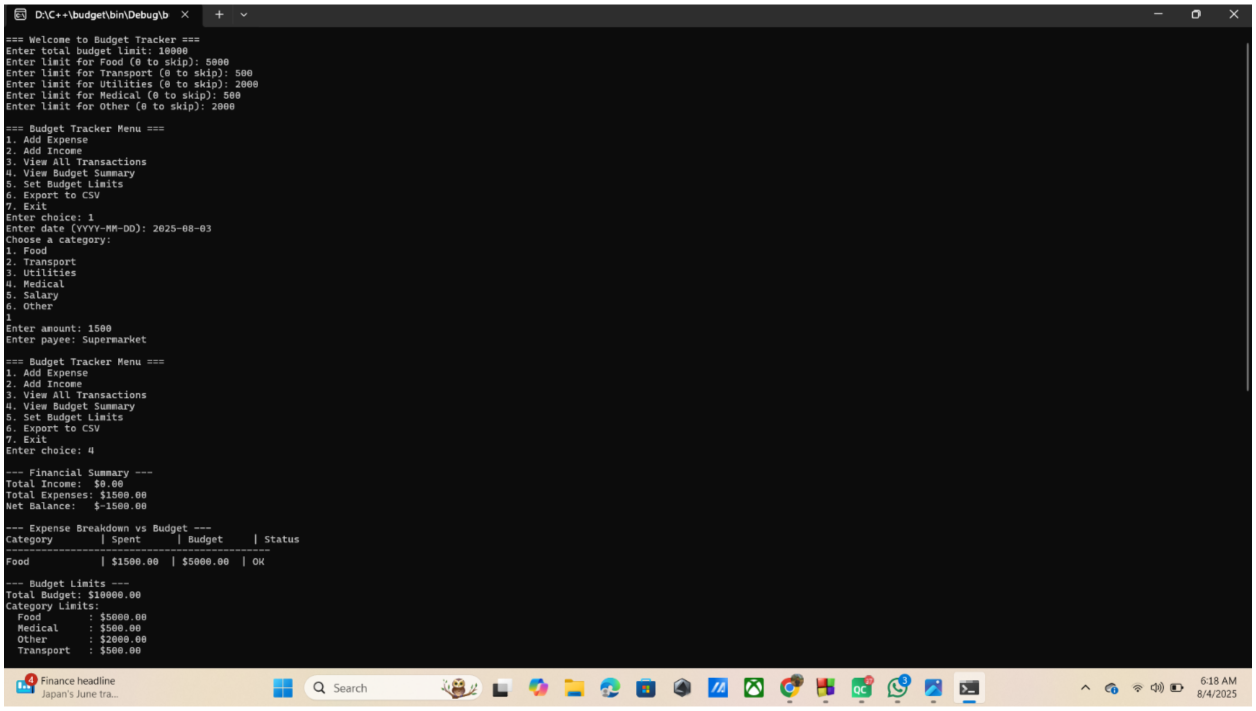
```
    transactions.push_back(move(transaction));
}

void displayAllTransactions() const {
    cout << "\n--- All Transactions ---\n";
    for (const auto& t : transactions) {
        t->display();
    }
}

void showSummary(const Budget& budget) const {
    map<string, double> expensesByCategory;
    map<string, double> incomeByCategory;
    double totalExpenses = 0.0;
    double totalIncome = 0.0;

    for (const auto& t : transactions) {
        if (t->isExpense()) {
            expensesByCategory[t->getCategory()] += t->getAmount();
            totalExpenses += t->getAmount();
        } else {
            incomeByCategory[t->getCategory()] += t->getAmount();
            totalIncome += t->getAmount();
        }
    }

    cout << "\n--- Financial Summary ---\n";
    cout << "Total Income:  $" << fixed << setprecision(2) << totalIncome << "\n";
    cout << "Total Expenses: $" << totalExpenses << "\n";
    cout << "Net Balance:   $" << (totalIncome - totalExpenses) << "\n\n";

    cout << "--- Expense Breakdown vs Budget ---\n";
    cout << left << setw(15) << "Category" << " | "
         << setw(10) << "Spent" << " | "
         << setw(10) << "Budget" << " | Status\n";
    cout << string(45, '-') << "\n";

    for (const auto& pair : expensesByCategory) {
  string category = pair.first;
  double amount = pair.second;
  double limit = budget.getCategoryLimit(category);
  string status = budget.isWithinCategoryLimit(category, amount) ? "OK" : "OVER";
  cout << setw(15) << category << " | $"
       << setw(8) << amount << " | $"
       << setw(8) << limit << " | " << status << "\n";
}
```

```
}

void exportToCSV(const string& filename) const {
    ofstream file(filename);
    if (!file.is_open()) {
        throw runtime_error("Could not open file for writing");
    }

    file << "Type,Date,Category,Amount,Detail\n";
    for (const auto& t : transactions) {
        file << (t->isExpense() ? "Expense" : "Income") << ","
             << t->getDate() << ","
             << t->getCategory() << ","
             << t->getAmount() << ",";

        if (auto expense = dynamic_cast<Expense*>(t.get())) {
            file << "Payee:" << expense->getPayee();
        } else if (auto income = dynamic_cast<Income*>(t.get())) {
            file << "Source:" << income->getSource();
        }

        file << "\n";
        }
        file.close();
    }
};

// Display menu
void displayMenu() {
    cout << "\n=== Budget Tracker Menu ===\n";
    cout << "1. Add Expense\n";
    cout << "2. Add Income\n";
    cout << "3. View All Transactions\n";
    cout << "4. View Budget Summary\n";
    cout << "5. Set Budget Limits\n";
    cout << "6. Export to CSV\n";
    cout << "7. Exit\n";
    cout << "Enter choice: ";
}

// Select category safely
string chooseCategory(bool forExpense = true) {
    cout << "Choose a category:\n";
    for (size_t i = 0; i < CategoryNames.size(); ++i) {
        if (!forExpense && CategoryNames[i] == "Salary") continue;
        cout << i + 1 << ". " << CategoryNames[i] << "\n";
    }
    int choice;
    cin >> choice;
    if (choice < 1 || choice > CategoryNames.size()) return "Other";
    return CategoryNames[choice - 1];
}

// Create Expense
unique_ptr<Transaction> createExpense() {
    string date, payee;
    double amount;

    cout << "Enter date (YYYY-MM-DD): ";
    cin >> date;
    string category = chooseCategory(true);
    cout << "Enter amount: ";
    cin >> amount;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << "Enter payee: ";
    getline(cin, payee);

    return make_unique<Expense>(date, category, amount, payee);
```

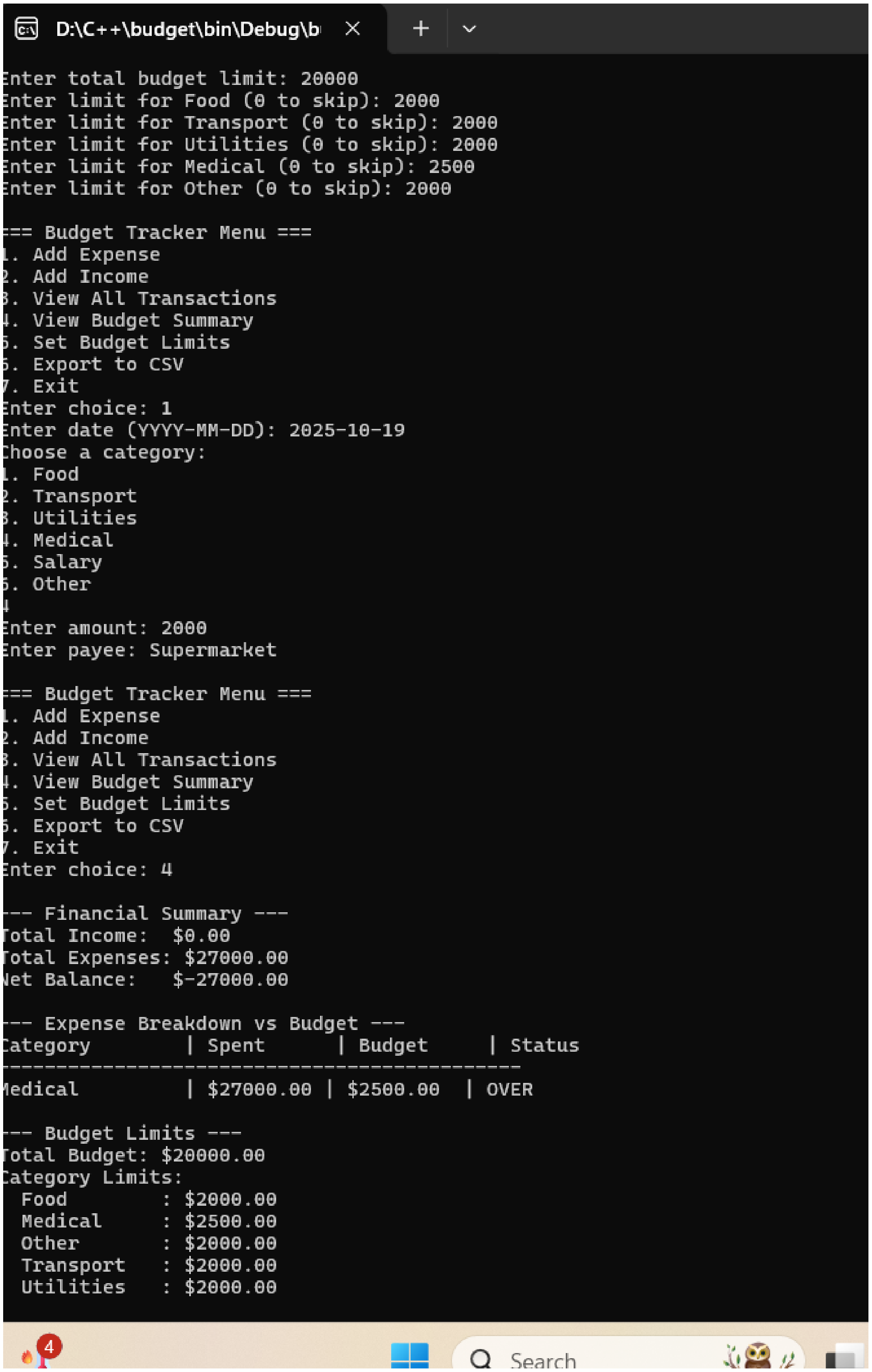
```
}

// Create Income
unique_ptr<Transaction> createIncome() {
    string date, source;
    double amount;

    cout << "Enter date (YYYY-MM-DD): ";
    cin >> date;
    string category = chooseCategory(false);
    cout << "Enter amount: ";
    cin >> amount;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << "Enter source: ";
    getline(cin, source);

    return make_unique<Income>(date, category, amount, source);
}

// Setup budget
void setupBudget(Budget& budget) {
    double totalLimit;
    cout << "Enter total budget limit: ";
    cin >> totalLimit;
    budget.setTotalLimit(totalLimit);

    for (const auto& category : CategoryNames) {
        if (category == "Salary") continue; // Income category
        double limit;
        cout << "Enter limit for " << category << " (0 to skip): ";
        cin >> limit;
        if (limit > 0) {
            budget.setCategoryLimit(category, limit);
        }
    }
}

// Main function
int main() {
    Tracker tracker;
    Budget budget;
    int choice;

    cout << "=== Welcome to Budget Tracker ===\n";
    setupBudget(budget);
```

```
while (true) {
    displayMenu();
    cin >> choice;

    try {
        switch (choice) {
            case 1:
                tracker.addTransaction(createExpense());
                break;
            case 2:
                tracker.addTransaction(createIncome());
                break;
            case 3:
                tracker.displayAllTransactions();
                break;
            case 4:
                tracker.showSummary(budget);
                budget.display();
                break;
            case 5:
                setupBudget(budget);
                break;
            case 6:
                    tracker.exportToCSV("transactions.csv");
                    cout << "Exported to transactions.csv\n";
                    break;
                case 7:
                    cout << "Exiting...\n";
                    return 0;
                default:
                    cout << "Invalid choice. Try again.\n";
            }
        } catch (const exception& e) {
            cerr << "Error: " << e.what() << "\n";
        }
    }
}
```

Input and Output:



```
=== Welcome to Budget Tracker ===
Enter total budget limit: 10000
Enter limit for Food (0 to skip): 5000
Enter limit for Transport (0 to skip): 500
Enter limit for Utilities (0 to skip): 2000
Enter limit for Medical (0 to skip): 500
Enter limit for Other (0 to skip): 2000

=== Budget Tracker Menu ===
1. Add Expense
2. Add Income
3. View All Transactions
4. View Budget Summary
5. Set Budget Limits
6. Export to CSV
7. Exit
Enter choice: 1
Enter date (YYYY-MM-DD): 2025-08-03
Choose a category:
1. Food
2. Transport
3. Utilities
4. Medical
5. Salary
6. Other
1
Enter amount: 1500
Enter payee: Supermarket

=== Budget Tracker Menu ===
1. Add Expense
2. Add Income
3. View All Transactions
4. View Budget Summary
5. Set Budget Limits
6. Export to CSV
7. Exit
Enter choice: 4

--- Financial Summary ---
Total Income:  $0.00
Total Expenses: $1500.00
Net Balance:   $-1500.00

--- Expense Breakdown vs Budget ---
Category        | Spent       | Budget      | Status
---------------------------------------------
Food            | $1500.00   | $5000.00   | OK

--- Budget Limits ---
Total Budget: $10000.00
Category Limits:
  Food        : $5000.00
  Medical     : $500.00
  Other       : $2000.00
  Transport   : $500.00
```



```
nter total budget limit: 20000
nter limit for Food (0 to skip): 2000
nter limit for Transport (0 to skip): 2000
nter limit for Utilities (0 to skip): 2000
nter limit for Medical (0 to skip): 2500
nter limit for Other (0 to skip): 2000

== Budget Tracker Menu ===
. Add Expense
. Add Income
. View All Transactions
. View Budget Summary
. Set Budget Limits
. Export to CSV
. Exit
nter choice: 1
nter date (YYYY-MM-DD): 2025-10-19
hoose a category:
. Food
. Transport
. Utilities
. Medical
. Salary
. Other

nter amount: 2000
nter payee: Supermarket

== Budget Tracker Menu ===
. Add Expense
. Add Income
. View All Transactions
. View Budget Summary
. Set Budget Limits
. Export to CSV
. Exit
nter choice: 4

-- Financial Summary ---
otal Income:   $0.00
otal Expenses: $27000.00
et Balance:    $-27000.00

-- Expense Breakdown vs Budget ---
ategory        | Spent     | Budget     | Status
---------------------------------------------
edical         | $27000.00 | $2500.00   | OVER

-- Budget Limits ---
otal Budget: $20000.00
ategory Limits:
 Food        : $2000.00
 Medical     : $2500.00
 Other       : $2000.00
 Transport   : $2000.00
 Utilities   : $2000.00
```