EXPENSE TRACKER SYSTEM

A C++ Object-Oriented Programming Project Report

INTRODUCTION

The **Expense Tracker System** is a console-based application developed in C++ to demonstrate the practical implementation of Object-Oriented Programming (OOP) concepts. The project focuses on automating the process of recording, organizing, and analyzing daily expenses in a simplified yet efficient manner. In the modern digital world, financial management is one of the most crucial aspects of an individual's routine life. People often struggle to maintain accurate records of their expenses, and manual tracking through notebooks or spreadsheets tends to be time-consuming, prone to errors, and difficult to update consistently. This project aims to provide a structured, user-friendly, and reliable solution that automates expense tracking using the powerful features of C++ and OOP.

The system allows users to add expenses under various predefined categories such as Food, Travel, and Utilities. It provides an option to view all recorded transactions and calculate total expenditures in real time. Each expense is represented as an object that encapsulates details such as category, amount, and date. The concept of inheritance is used to differentiate between various expense types, while polymorphism is employed to ensure that each expense type displays information specific to its category. Constructors initialize object data upon creation, while destructors manage memory cleanup efficiently when the program terminates. Dynamic memory allocation ensures flexibility in object creation during runtime, allowing the system to handle a varying number of expenses depending on user input. Overall, the Expense Tracker System stands as a simple yet elegant demonstration of how OOP principles can be harnessed to develop a useful and educational software tool.

PURPOSE OF THE PROJECT

The primary purpose of the Expense Tracker System is to simplify and automate the management of daily financial transactions. Traditionally, people have relied on manual record-keeping or spreadsheets to track expenses. While these methods can serve basic needs, they often become cumbersome as the number of transactions grows. Errors, omissions, and disorganization are common in such systems, especially when users need to retrieve specific data or analyze their spending patterns. This project addresses these limitations by creating an interactive and automated system that performs these tasks effortlessly and accurately.

In addition to its practical benefits, the project serves as a valuable learning exercise for understanding the real-world application of Object-Oriented Programming concepts in C++. Each class in the system represents a specific entity—such as an expense, a category, or a record manager—illustrating how encapsulation and abstraction make code more structured and reusable. By segregating different functionalities into classes and utilizing inheritance, the program achieves a high level of modularity, making it easy to extend or modify in the future. The system not only helps users organize their personal finances but also acts as an educational model for students to understand how complex data operations can be represented in an object-oriented framework.

Moreover, the purpose of this project extends beyond personal finance management. It demonstrates how software can bring order, accuracy, and efficiency to tasks that are otherwise repetitive and prone to human error. By automating expense calculations and providing instant summaries, this project highlights the potential of programming in improving everyday life activities through intelligent automation.

ABSTRACT

The Expense Tracker System is a mini-project developed in C++ to illustrate how Object-Oriented Programming principles can be applied to solve real-life problems in an efficient and modular way. The project focuses on financial record management and provides a console-based interface where users can easily record, view, and analyze their daily expenses. Each expense is associated with a category—such as Food, Travel, or Utilities—and stored dynamically during program execution. The program applies inheritance to create specialized expense categories, and polymorphism allows each category to display unique information according to its type.

The system consists of a base class called Expense, which holds the fundamental properties common to all expenses, such as category, amount, and date. Derived classes like FoodExpense, TravelExpense, and UtilityExpense extend this base class to represent specific categories. The concept of virtual functions is employed to achieve runtime polymorphism, allowing the program to call the appropriate display function depending on the actual object type. The ExpenseManager class acts as the central controller that manages all expense records, performs addition and deletion of entries, and calculates total expenditure. The project also utilizes constructors to initialize data automatically upon object creation and destructors to release memory once the objects are no longer needed.

Through this system, users can experience a structured approach to managing their daily expenses while learners can observe the seamless integration of OOP principles in practice. It bridges the gap between theoretical understanding and practical implementation, offering an educational yet functional example of software design.

SYSTEM CONFIGURATION

Hardware Requirements

Component	Minimum Specification	Recommended Specification
RAM	**2 GB**	**4 GB**
Processor	**Dual Core**	**Intel i3 or above**
Hard Disk	**500 MB Free**	**1 GB Free**
Display	**Standard Console**	**HD Console Terminal**
Input Device	**Keyboard**	**Keyboard**

The Expense Tracker System is a lightweight and portable program that can run on most modern computers without requiring special hardware or additional software dependencies. The project was designed and tested on the Windows operating system, but it is equally compatible with Linux and macOS environments. It utilizes only the standard C++ libraries such as <iostream>, <string>, and <vector>, ensuring platform independence and smooth execution.

The minimum hardware requirements include 2 GB of RAM, a dual-core processor, and 500 MB of free storage space. However, for optimal performance, a system with at least 4 GB of RAM and a modern Intel or AMD processor is recommended. The program runs entirely on the command-line interface, so no graphical hardware acceleration is needed. The software requirements include a standard C++ compiler such as GCC, Turbo C++, Dev C++, or Code::Blocks. Since the project uses only basic console input and output operations, it consumes minimal system resources and can execute efficiently even on older computers.

This configuration ensures that students and developers can easily compile, run, and test the program on any available system without encountering compatibility issues. Its simplicity makes it an ideal candidate for academic projects and practical demonstrations of OOP principles.

SYSTEM DESIGN

The design of the Expense Tracker System follows a modular structure, which ensures that each part of the system performs a specific and well-defined task. The overall architecture is based on the interaction between objects that represent real-world entities such as expenses, categories, and management functions. The base class Expense acts as the foundation for defining all types of expenses, containing common attributes like category name, amount, and date. Derived classes—FoodExpense, TravelExpense, and UtilityExpense—extend the base class, each representing a distinct expense type. The inheritance structure allows these derived classes to override the display function to present data in a category-specific format, which demonstrates polymorphism in action.

The ExpenseManager class serves as the central control unit of the program. It maintains a vector of pointers to Expense objects, allowing the program to store any type of expense dynamically at runtime. The manager class includes functions for adding new expenses, displaying all recorded entries, and computing the total amount spent. When the user exits the program, the destructor of the ExpenseManager class ensures that all dynamically allocated memory is released properly, demonstrating responsible memory management.

This design structure is intentionally simple yet highly extensible. Future developers can easily add new expense categories by creating additional derived classes without modifying the core program logic. Such flexibility is one of the key advantages of OOP-based design, promoting scalability and maintainability in software development.

ALGORITHM AND FLOW OF EXECUTION

Algorithm Steps

1. **Start program.**
2. **Display main menu.**
3. **Accept user choice.**
4. **If the choice is "Add Expense", ask for amount and date, then create appropriate expense object dynamically.**
5. **Add object to the expense vector.**
6. **If the choice is "View Expenses", iterate through the vector and display all expenses.**
7. **If "View Total", calculate and display total.**
8. **If "Exit", delete all allocated objects and close the program.**
9. **End.**

The flow of the Expense Tracker System begins with the program's main function, which presents a user-friendly text menu on the console. The user is prompted to choose from a list of options such as adding a new expense, viewing existing expenses, calculating total expenditure, or exiting the program. If the user opts to add an expense, the program requests the amount and date, and then dynamically creates an object of the corresponding expense class based on the chosen category. This object is then added to the vector of expenses maintained by the ExpenseManager class.

When the user chooses to view all expenses, the program iterates through the vector and calls the virtual display function for each object. Thanks to polymorphism, the correct function for each derived class is invoked, ensuring category-specific display messages. Similarly, the total expense calculation is performed by summing the amounts of all objects in the vector. When the user decides to exit, the destructor of the ExpenseManager class automatically releases the memory occupied by the dynamically created expense objects, confirming the proper use of destructors in C++.

Program:

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// Base Class: Expense
class Expense {
protected:
    string category;
    double amount;
    string date;

public:
    Expense(string c = "", double a = 0.0, string d = "")
        : category(c), amount(a), date(d) {}

    virtual void displayExpense() {
        cout << category << " Expense: Rs." << amount << " on " << date << endl;
    }

    double getAmount() { return amount; }
    string getCategory() { return category; }

    virtual ~Expense() {
        // Destructor for cleanup
    }
};
```

```
// Derived Classes
class FoodExpense : public Expense {
public:
    FoodExpense(double a, string d) : Expense("Food", a, d) {}
    void displayExpense() override {
        cout << "🍔 Food Expense: Rs." << amount << " on " << date << endl;
    }
};

class TravelExpense : public Expense {
public:
    TravelExpense(double a, string d) : Expense("Travel", a, d) {}
    void displayExpense() override {
        cout << "🚗 Travel Expense: Rs." << amount << " on " << date << endl;
    }
};

class UtilityExpense : public Expense {
public:
    UtilityExpense(double a, string d) : Expense("Utilities", a, d) {}
    void displayExpense() override {
        cout << "💡 Utility Expense: Rs." << amount << " on " << date << endl;
    }
};

// Manager Class
class ExpenseManager {
```

```
    vector<Expense*> expenses;

public:
    void addExpense(Expense* e) {
        expenses.push_back(e);
        cout << "✅ Expense added successfully!\n";
    }

    void showAllExpenses() {
        cout << "\n----- All Recorded Expenses -----\n";
        for (auto e : expenses)
            e->displayExpense();
    }

    void showTotal() {
        double total = 0;
        for (auto e : expenses)
            total += e->getAmount();
        cout << "\n💰 Total Expenses: Rs." << total << endl;
    }

    ~ExpenseManager() {
        for (auto e : expenses)
            delete e;
        cout << "\nMemory cleared. Thank you for using Expense Tracker!\n";
    }
};
```

```
// Main Function
int main() {
    ExpenseManager manager;
    int choice;

    do {
        cout << "\n========== EXPENSE TRACKER SYSTEM ==========\n";
        cout << "1. Add Food Expense\n";
        cout << "2. Add Travel Expense\n";
        cout << "3. Add Utility Expense\n";
        cout << "4. View All Expenses\n";
        cout << "5. View Total Expenses\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice >= 1 && choice <= 3) {
            double amt;
            string date;
            cout << "Enter amount (Rs.): ";
            cin >> amt;
            cout << "Enter date (DD/MM/YYYY): ";
            cin >> date;

            switch (choice) {
                case 1: manager.addExpense(new FoodExpense(amt, date)); break;
                case 2: manager.addExpense(new TravelExpense(amt, date)); break;
                case 3: manager.addExpense(new UtilityExpense(amt, date)); break;
```

```
            }
        } else if (choice == 4) {
            manager.showAllExpenses();
        } else if (choice == 5) {
            manager.showTotal();
        } else if (choice == 6) {
            cout << "Exiting... Goodbye!\n";
        } else {
            cout << "Invalid choice! Try again.\n";
        }

    } while (choice != 6);

    return 0;
}
```

Output

```
========== EXPENSE TRACKER SYSTEM ==========
1. Add Food Expense
2. Add Travel Expense
3. Add Utility Expense
4. View All Expenses
5. View Total Expenses
6. Exit
Enter your choice: 1
Enter amount (Rs.): 250
Enter date (DD/MM/YYYY): 1/11/2025
✅ Expense added successfully!
```

```
========== EXPENSE TRACKER SYSTEM ==========
1. Add Food Expense
2. Add Travel Expense
3. Add Utility Expense
4. View All Expenses
5. View Total Expenses
6. Exit
Enter your choice: 2
Enter amount (Rs.): 100
Enter date (DD/MM/YYYY): 1/11/2025
✅ Expense added successfully!
```

```
========== EXPENSE TRACKER SYSTEM ==========
1. Add Food Expense
2. Add Travel Expense
3. Add Utility Expense
4. View All Expenses
5. View Total Expenses
6. Exit
Enter your choice: 4

----- All Recorded Expenses -----
🍔 Food Expense: Rs.250 on 1/11/2025
🚗 Travel Expense: Rs.100 on 1/11/2025
```

```
========== EXPENSE TRACKER SYSTEM ==========
1. Add Food Expense
2. Add Travel Expense
3. Add Utility Expense
4. View All Expenses
5. View Total Expenses
6. Exit
Enter your choice: 5

💰 Total Expenses: Rs.350
```

ANALYSIS OF OUTPUT

When executed, the Expense Tracker System provides a smooth, text-based interface that interacts with the user through simple menu-driven options. The user can easily add multiple expenses under different categories and view the details at any point during the session. The output displays each expense in a clean and organized manner, indicating the category, amount, and date of transaction. For instance, when a user adds a food expense of Rs.250 dated 02/11/2025, the program confirms the successful addition of the entry. Similarly, when the "View All Expenses" option is selected, all recorded expenses are displayed neatly on the console with the corresponding emojis and formatting that make the interface visually appealing despite being text-based.

The results confirm the correct implementation of core OOP principles. Each time an expense is entered, the corresponding derived class object is dynamically created and stored. The correct display function is invoked for each category, validating the use of polymorphism. The total expense calculation works accurately by summing all stored amounts. Additionally, when the program ends, destructors execute automatically to clear memory, demonstrating efficient resource management.

FEATURES AND BENEFITS

The Expense Tracker System provides several valuable features. It allows categorized expense tracking, real-time total calculations, and organized storage of all entries. The design is modular, flexible, and easy to extend. Each class performs a distinct role, making the code clean and maintainable. The console interface ensures simplicity while keeping the focus on programming logic rather than interface complexity.

From an academic perspective, this project effectively demonstrates the practical applications of encapsulation, inheritance, constructors, destructors, and polymorphism. It encourages logical thinking and reinforces students' understanding of OOP design principles. Additionally, the project's real-world relevance makes it more engaging and meaningful than abstract programming exercises.

LIMITATIONS

Despite its functionality, the Expense Tracker System has certain limitations. It operates entirely in a console environment and lacks a graphical user interface. Data entered by the user is not stored permanently; once the program is closed, all recorded expenses are lost since file handling has not been implemented. Furthermore, the number of expense categories is fixed, restricting flexibility. The program also lacks advanced analytical features such as monthly or yearly reports, graphical representations, and data export options.

These limitations, however, provide a foundation for future enhancement, offering students opportunities to expand the system further by incorporating additional programming techniques.

Conclusion

The Expense Tracker System successfully demonstrates the application of Object-Oriented Programming (OOP) concepts in developing a structured, modular, and efficient software solution using C++. Through this project, the abstract principles of OOP—such as encapsulation, inheritance, polymorphism, and abstraction—are not merely presented as theoretical ideas but are brought to life through practical implementation. Each component of the program contributes to an integrated system that automates the process of tracking and analyzing personal financial data, offering both educational and practical value.

From a functional standpoint, the system provides users with a simple and effective means to manage their daily expenses. By categorizing expenditures into predefined groups such as Food, Travel, and Utilities, it helps users gain a clearer picture of their spending habits. The ability to calculate total expenses instantly allows users to make informed decisions about budgeting and saving. Although the project is designed as a console-based application, its efficiency and logical structure reflect the potential of object-oriented software to simplify everyday tasks that people often manage manually.

The project's development process reinforces a deeper understanding of OOP design methodology. The Expense base class serves as the foundation that defines the shared structure of all expense objects, ensuring consistency in data representation. Derived classes such as FoodExpense, TravelExpense, and UtilityExpense extend this structure, showcasing inheritance and demonstrating how new types can be introduced without altering existing code—a core principle of reusability in OOP. Polymorphism plays a vital role in making the system dynamic and flexible, allowing the program to determine which display function to execute based on the specific expense type at runtime. This feature exemplifies how C++ can handle complex relationships between classes elegantly and efficiently.Furthermore, the use of constructors and destructors illustrates the importance of resource management in software development. Constructors are responsible for initializing expense details upon object creation, ensuring that each record starts in a well-defined state. Destructors, on the other hand, take care of freeing memory when the program terminates, demonstrating clean programming practices that prevent memory leaks—a critical consideration in C++ applications.

REFERENCES:

1. Programming in C++ by E. Balagurusamy
2. Object-Oriented Programming with C++ by Robert Lafore
3. Official C++ Documentation – https://cplusplus.com4
4. TutorialsPoint – https://www.tutorialspoint.com/cplusplus
5. GeeksforGeeks – https://www.geeksforgeeks.org/oops-concepts-in-cpp