Group Project: Cross-Language Development

Date: April 06, 2025

- **Term:** Spring 2025 Second Bi-term
- **Class:** Advanced Programming Languages -MSCS-632-B01
- **Instructor:** Dr. Dax Bradley
- **University of the Cumberlands**

The GitHub repository, https://github.com/baralsamrat/MSCS632_Project_Group, contains the source code for the C++ and Python expense tracker applications, along with the metrics analysis script and this report. The repository's commit history reflects the individual contributions of each team member.

Team Contributions: Group 1

Name	Email	Contributions
Prasanna Adhikari	padhikari34605@ucumberlands.edu	Led the C++ implementation, focusing on memory management with smart pointers and efficient use of STL containers.
Shashwat Baral	sbaral29114@ucumberlands.edu	Developed the Python implementation, emphasizing the use of dynamic typing and the simplicity of Python's data structures.
Samrat Baral	sbaral30064@ucumberlands.edu	Implemented the metrics and visualization components, providing side-by-side comparisons of code metrics between the two languages.
Sahithi Bontha	sbontha35464@ucumberlands.edu	Coordinated integration, documentation, and testing, ensuring all contributions were reflected in the GitHub repository.

Comparison of Expense Tracker Implementations: C++ vs Python

Introduction

This report compares two implementations of an Expense Tracker Application—one in C++ and one in Python. Both versions support the same core functions: adding expenses, filtering expenses by date or category, and displaying expense summaries. However, they leverage language-specific features that have different impacts on design, performance, and readability.

- Add expenses with fields: Date, Amount, Category, and Description.
- Filter expenses by date range or category.
- View total expenses by category and overall.

- Compare implementations between Python and C++.

Both files are analyzed for key metrics that reflect the implementation of the core features (e.g., functions for filtering, summarizing, etc.).

Getting Started

```
chmod +x main.sh
./main.sh
```

Screenshot

https://github.com/baralsamrat/MSCS632_Project_Group/blob/main/screenshots/1/Figure_1.png

---

Key Language-Specific Features

Below is a table comparing three major language-specific features—memory management, data structures, and error handling/verbosity—with code snippets to showcase the differences between the C++ and Python implementations.

Feature	C++ Implementation	Python Implementation
Memory Management	**Approach:** Uses smart pointers (`std::unique_ptr`) to ensure deterministic memory cleanup.	**Approach:** Relies on automatic garbage collection, simplifying the code without manual memory management.

	Code Snippet:
	void addExpense(const std::string &date, double amount, const std::string &category, const std::string &description) {
	expenses.push_back(std::make_unique(Expense{date, amount, category, description}));
	}
		Code Snippet:
		def add_expense(date, amount, category, description):
		expense = {
		"date": date,
		"amount": amount,
		"category": category,
		"description": description
		}
		expenses.append(expense)

Data Structures	**Approach:** Defines a `struct Expense` for type safety and uses STL containers (`std::vector` and `std::map`) for efficient storage and retrieval.	**Approach:** Uses dynamic data structures like lists and dictionaries, which reduce boilerplate code and enhance readability.

Feature	C++ Implementation	Python Implementation

	Code Snippet:
	struct Expense {
	std::string date;
	double amount;
	std::string category;
	std::string description;
	};
	
	std::vector<std::unique_ptr> expenses;
		Code Snippet:
		expenses = []
		
		def add_expense(date, amount, category, description):
		expense = {
		"date": date,
		"amount": amount,
		"category": category,
		"description": description
		}
		expenses.append(expense)

Error Handling and Code Verbosity	**Approach:** C++ requires explicit error handling and type declarations, resulting in more verbose but predictable code.	**Approach:** Python's concise syntax and exception handling allow for faster prototyping and easier maintenance, though with less control.
	Discussion: Explicit type checking and manual error handling help catch issues at compile time.	**Discussion:** Dynamic typing and built-in exceptions simplify development at the expense of potentially catching errors later (at runtime).
Standard Library	**C++ Standard Template Library (STL)** provides powerful data structures and algorithms. Offers high performance but requires understanding of templates and iterators	**Python's standard library** is extensive and provides a wide range of modules for various tasks. Offers a high level of abstraction and ease of use.

	Code Snippet:
	#include iostream
	#include vector
	#include algorithm
	int main()
	{
	std::vector numbers = {3, 1, 4, 1, 5};
	std::sort(numbers.begin(), numbers.end());
	for (int num : numbers) {
	std::cout << num << " ";
	};
	std::cout << std::endl;
	return 0;
	}
		Code Snippet:
		numbers = [3, 1, 4, 1, 5]
		numbers.sort()
		print(numbers))

Impact on Design, Performance, and Readability

- **Design:**
  The C++ implementation's explicit memory management and static type system demand a disciplined design approach, which can result in highly optimized and robust applications. In contrast, Python's dynamic typing and high-level abstractions allow for rapid development and simpler designs.

- **Performance:**
  C++ typically outperforms Python in raw execution speed and resource management due to its low-level control and compile-time optimizations. Python, while generally slower, is more than adequate for applications where rapid development and maintainability are prioritized.

- **Readability:**
  Python's concise and expressive syntax makes the code more accessible, particularly for those new to programming. C++ code tends to be more verbose, which can improve clarity around resource management but might be more challenging for quick prototyping.

Conclusion

Both the C++ and Python implementations of the Expense Tracker meet the core functional requirements, yet each language's unique features influence the application in different ways. The C++ version benefits from tight control over memory and high performance, while the Python version excels in readability and rapid development. This comparative study underscores the trade-offs inherent in choosing one language over another and demonstrates that both approaches can successfully implement the same core functionality with distinct advantages.

C++ and Python offer distinct advantages and disadvantages in terms of design, performance, and readability. C++ excels in performance-critical applications with its fine-grained control and static typing. Python prioritizes ease of use and rapid development with automatic memory management and a rich standard library. While it may have performance limitations, its concise syntax and readability make it suitable for a wide range of applications.