Currency Converter

CCP Report

PF (CT-175)

Group Name:

ZenCoders

Submitted By:

- Muhammad Tayyab Khan (CT-25088)
- Muhammad Ayaan (CT-24288)
- Vishal Kumar (CT-25081)

Teachers:

Sir Abdullah

Index

Table of Contents

Code Analysis & Usage

1. Code Understanding

The core of this program is its scalability, which is achieved through two key C concepts: structs and FILE handling.

- **struct Currency:** A struct is used as a "blueprint" to hold all the data for a single currency (code, name, and rate). This is far more organized than using separate arrays.
- **FILE* and fopen():** On startup, the program opens rates.txt. This external file holds all 180+ currency rates. This means we can add, remove, or update currency rates without *ever* recompiling the C code.
- **fscanf() Loop:** The program reads the rates.txt file line-by-line using fscanf inside a while loop. This loop automatically loads all currencies into an array of structs (struct Currency currencies[200]).
- **Base-USD Logic:** The program does not need 32,000+ conversion pairs. It uses a single base currency (USD). To convert PKR to EUR, it first converts (PKR -> USD), and then (USD -> EUR). This two-step formula works for all currencies.

2. Program Usage

The program is a simple, menu-driven console application:

- **Menu 1 (Convert):** The user provides an amount, a "from" code (e.g., PKR), and a "to" code (e.g., USD). The program finds the rates and prints the result. It also handles user errors (like "pkr") by converting all input to uppercase.
- **Menu 2 (List Currencies):** This prints all 180+ currencies loaded from the rates.txt file, so the user knows which codes are available to use.
- **Menu 3 (Exit):** This safely closes the program.

Code Improvement

This C project is a strong foundation that successfully meets all core requirements. However, there are several ways this program could be improved and expanded in the future.

1. Graphical User Interface (GUI)

The current program runs in a text-based console. The most significant improvement would be to build a GUI.

- **Method:** This could be done in C using a library like GTK+, or by porting the logic to C++ (Qt) or Python (Tkinter).
- **Benefit:** A GUI would be far more user-friendly, replacing text-based code entry (e.g., "PKR") with drop-down menus, which would eliminate user error.

2. Live Exchange Rates via API

The current rates.txt file is static and must be updated manually. A more advanced version would connect to the internet to get live data.

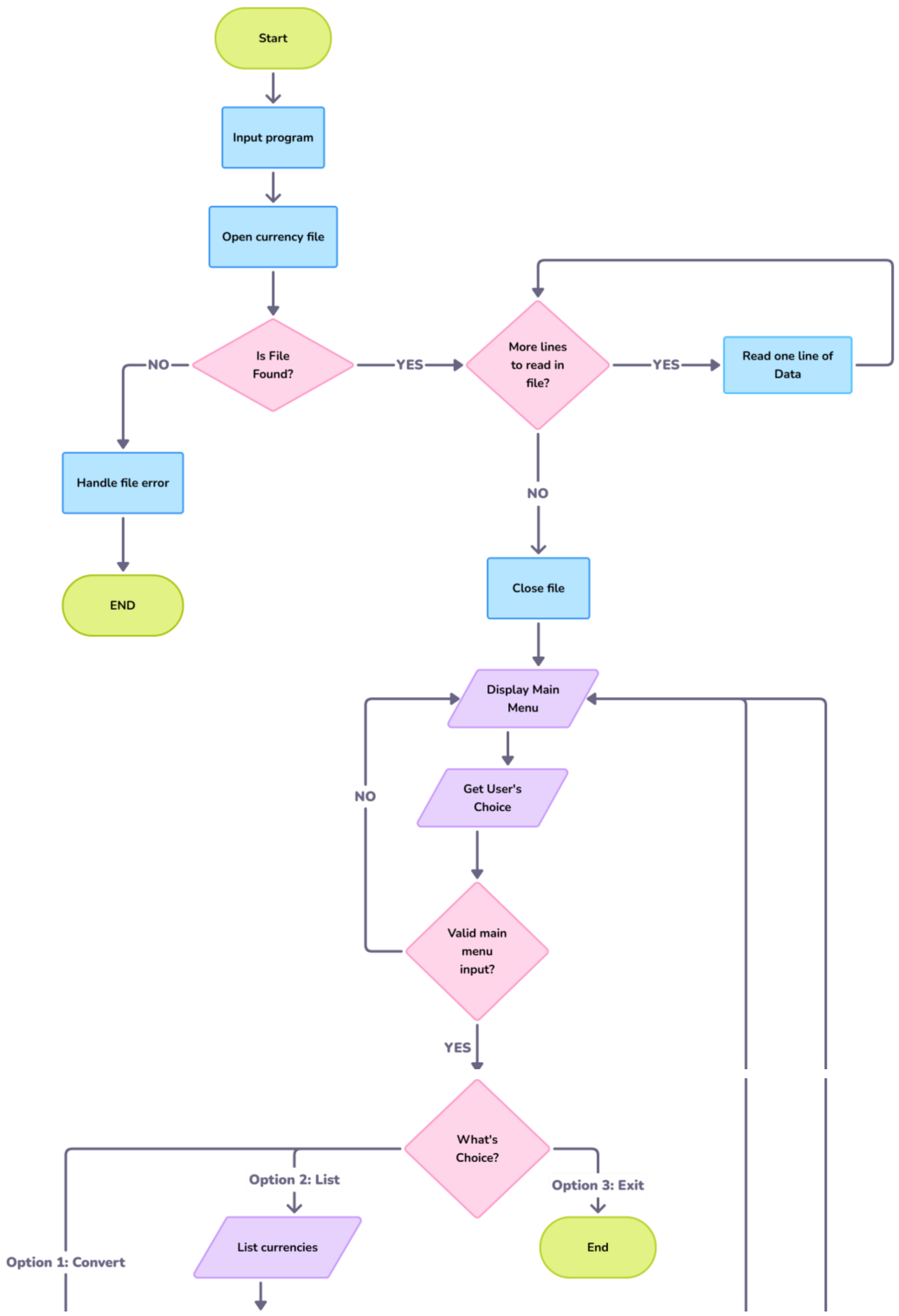
- **Method:** This would require using a library like libcurl in C to make an HTTP request to a free currency API (like ExchangeRate-API). The program would parse the (JSON) response to get the latest rates every time it starts.
- **Benefit:** This would make the converter accurate to the minute and would eliminate the need for any manual file management.

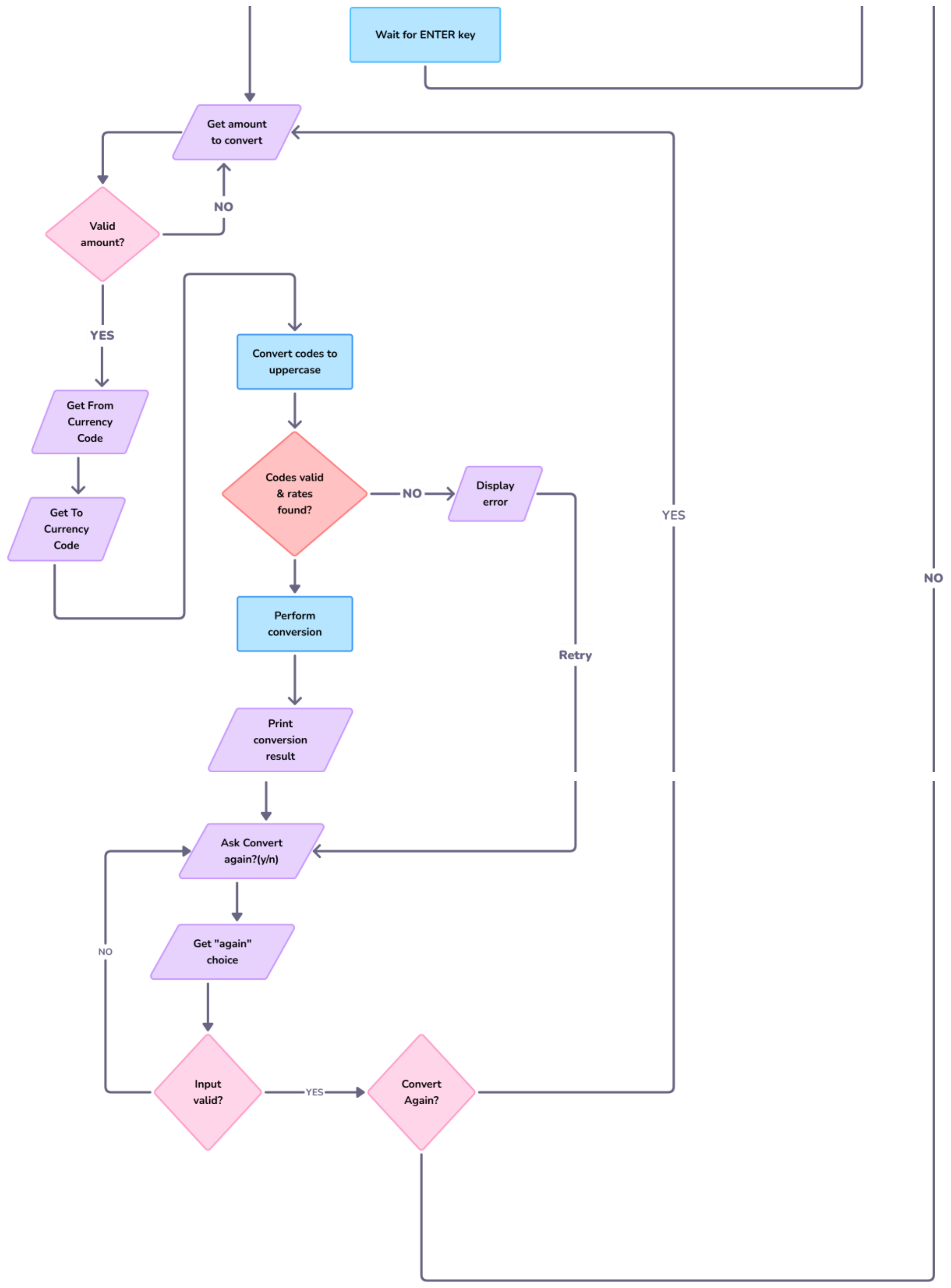
3. More Robust Error Handling

While the current code handles bad numerical input (e.g., typing "abc"), it could be even more robust.

- **Method:** We could add checks for negative numbers for the *amount* to convert, or add a search function within "List Currencies" so the user doesn't have to scroll through 180+ entries.

Program Flowchart




Wait for ENTER key
Get amount to convert
Valid amount?
NO
YES
Get From Currency Code
Get To Currency Code
Convert codes to uppercase
Codes valid & rates found?
NO
Display error
Retry
Perform conversion
Print conversion result
Ask Convert again?(y/n)
Get "again" choice
NO
Input valid?
YES
Convert Again?
YES
NO

Conclusion

This project was a comprehensive and practical exercise in C programming that went far beyond basic syntax. As **ZenCoders**, our group successfully designed and built a console application that is not just functional, but also robust, scalable, and efficient—principles that are critical in professional software development.

The most important takeaway was learning to **separate logic from data**. By using `FILE` handling to read `rates.txt`, we created a program that can be updated (with new currencies or new rates) without ever touching or recompiling the C code. This is a powerful, real-world concept.

We gained direct experience with key C topics:

- **Data Structuring:** Using `structs` to organize data was a major lesson in efficiency, making the code far cleaner than managing parallel arrays.
- **File I/O:** We mastered `fopen`, `fscanf`, and `fclose`, including the critical step of checking for `NULL` to prevent crashes if the file is missing.
- **Robust Input:** We learned the hard way *why* `scanf` is tricky. Implementing the `if (scanf(...) != 1)` and `while(getchar() != '\n')` loops taught us the importance of handling bad user input and managing the input buffer to prevent bugs.
- **Algorithmic Logic:** Designing the two-step "Base-USD" conversion formula was a key insight. It simplified a problem that seemed to require 32,000+ calculations down to one single, reusable formula.

Ultimately, this project was a success. We achieved all our objectives and produced a final program that is complete, user-friendly, and demonstrates a strong, practical understanding of core C programming.