**Title: A Console-Based C++ Store**

**Abstract**

This report provides an in-depth analysis of the development of a console-based shopping application, named C++ Store, implemented using C++. The project aims to simulate a real-world shopping experience through a command-line interface, allowing users to browse various product categories, add items to their cart, and proceed with the purchase while capturing delivery details. Exception handling mechanisms are incorporated to enhance the robustness of the application. The report delves into the design, implementation, logic, and future improvements for the application, showcasing its educational value in understanding fundamental C++ programming concepts.

**Introduction**

The digital revolution has significantly influenced shopping behaviors, driving the need for robust and user-friendly e-commerce platforms. This project, C++ Store, is an educational endeavor to mimic an e-commerce application using the C++ programming language. The project not only serves as a practical demonstration of C++ capabilities but also emphasizes the importance of exception handling, user input validation, and file management in software development. By providing a console-based interface, the project simplifies the complexities of e-commerce applications, making it accessible for educational purposes.

**Objectives**

The primary objectives of the C++ Store project are:

* To develop a console-based shopping application using C++.
* To create a variety of product categories and functionalities for adding items to the shopping cart.
* To implement exception handling to manage errors effectively.
* To utilize file handling for storing user delivery details.

**Methodology**

The development of the C++ Store followed a structured methodology involving several key stages:

1. **Design**: Conceptualizing the application structure, including the product categories, user interface, and flow of operations.
2. **Development**: Writing and integrating C++ code to bring the design to life, with a focus on modular functions for each product category.
3. **Testing**: Rigorous testing of the application to ensure it handles user inputs and errors gracefully.
4. **Documentation**: Comprehensive documentation of the code, user guide, and this detailed report.

**Implementation Details**

The implementation of the C++ Store consists of several functions and modules that work together to create a seamless shopping experience. Below is a detailed explanation of the key components and workflows within the application.

**Loading Screen**

The application begins with a loading screen to simulate the initialization of a shopping platform. This screen displays a progress bar to engage the user during the brief startup period.

**Pseudo-code:**

*function loading\_screen()*

*display "LOADING"*

*for i from 1 to 69*

*print progress bar character*

*pause for 20 milliseconds*

*end for*

*end function*

function loading\_screen() display "LOADING" for i from 1 to 69 print progress bar character pause for 20 milliseconds end for end function

**Main Menu**

The main menu presents the user with various product categories. Each category corresponds to a function that handles the selection and addition of items to the cart.

**Pseudo-code:**

*function loading\_screen()*

*display "LOADING"*

*for i from 1 to 69*

*print progress bar character*

*pause for 20 milliseconds*

*end for*

*end function*

**Product Category Functions**

Each product category function prompts the user to select specific items and adds the selected items to the cart. Below is an example of the **instant\_foods** function, with similar structures for other categories.

**Pseudo-code for Instant Foods:**

*function instant\_foods()*

*display options for noodles, pasta, and chips*

*get user input for selection*

*switch selection*

*case 1: add noodles to cart*

*case 2: add pasta to cart*

*case 3: add chips to cart*

*default: handle invalid input*

*end switch*

*end function*

function instant\_foods() display options for noodles, pasta, and chips get user input for selection switch selection case 1: add noodles to cart case 2: add pasta to cart case 3: add chips to cart default: handle invalid input end switch end function

**Exception Handling**

Exception handling is integral to the application, ensuring that invalid inputs and file errors do not crash the program. The use of **try-catch** blocks helps manage exceptions effectively.

**Pseudo-code for User Input Validation:**

*function get\_user\_input()*

*try*

*get input from user*

*if input is invalid*

*throw exception "Invalid input"*

*end if*

*catch exception*

*display error message*

*clear input buffer*

*retry input*

*end try*

end functionfunction get\_user\_input() try get input from user if input is invalid throw exception "Invalid input" end if catch exception display error message clear input buffer retry input end try end function

**Pseudo-code for File Handling:**

*function add\_to\_file()*

*try*

*open file "delivery\_data.txt" in append mode*

*if file cannot be opened*

*throw exception "Error opening file"*

*end if*

*write user details to file*

*close file*

*catch exception*

*display error message*

*end try*

end functionfunction add\_to\_file() try open file "delivery\_data.txt" in append mode if file cannot be opened throw exception "Error opening file" end if write user details to file close file catch exception display error message end try end function

**User Delivery Details**

After the user completes their shopping, they are prompted to enter their delivery details, which are stored in a file. This ensures the persistence of user data for delivery purposes.

**Pseudo-code:**

*function capture\_delivery\_details()*

*get user name, address, and phone number*

*call add\_to\_file() with user details*

*end function*

function capture\_delivery\_details() get user name, address, and phone number call add\_to\_file() with user details end function

**Exception Handling**

The application incorporates exception handling to manage:

* **User Input Validation**: Ensuring that user inputs are valid and prompting the user to re-enter data in case of invalid inputs.
* **File Operations**: Managing file read/write errors to ensure data integrity and program stability.

**Example of User Input Validation**

In the product selection functions, user inputs are validated to ensure they fall within acceptable ranges. If an invalid input is detected, the input buffer is cleared, and the user is prompted to enter a valid choice.

**Example of File Handling**

When writing user delivery details to a file, the program checks if the file can be opened. If not, it throws an exception and displays an appropriate error message to the user.

**Results**

The C++ Store application successfully demonstrates the following:

* Users can browse and select products from various categories.
* The application accurately calculates and displays the total bill.
* User delivery details are captured and stored in a text file.
* Exception handling mechanisms ensure the application handles errors gracefully without crashing.

**Discussion**

The C++ Store project highlights several important aspects of software development in C++:

* **Modular Design**: The use of functions for different product categories and operations ensures a clean and maintainable codebase.
* **Exception Handling**: Proper error management enhances the robustness and user experience of the application.
* **File Handling**: Efficiently managing file operations to store user data demonstrates the practical use of file I/O in C++.

**Conclusion**

The C++ Store project successfully meets its objectives of creating a console-based shopping application with effective error management. It serves as an educational tool, demonstrating key C++ programming concepts and the importance of robust software design.

**Future Work**

Future improvements for the C++ Store application include:

* **Object-Oriented Programming (OOP)**: Refactoring the application using OOP principles to create a more structured and scalable design. For instance, creating classes for Product, Cart, and User to encapsulate related data and functionalities.
* **Enhanced User Interface**: Improving the console-based interface to provide a more user-friendly experience. This could include better navigation menus and detailed product descriptions.
* **Database Integration**: Integrating a database to manage products and user details more efficiently, allowing for dynamic updates and queries.
* **Additional Features**: Implementing features such as product search, discounts, user authentication, and order history to enhance the functionality and user experience of the application.