Project - Configurable Data Processing System in C++

Project Overview:

This project is a sophisticated C++ application designed to enhance your understanding of core C++ concepts, including Object-Oriented Programming (OOP), Standard Template Library (STL), Templates, File Handling, and Exception Handling. The application focuses on creating a dynamic and extensible system for processing different types of data based on user-provided configuration files.

Through this project, you will develop a configurable system that demonstrates the power of abstraction, polymorphism, and modular design while ensuring robust error handling and maintainability.

Key Features

1. **Dynamic Configuration Parsing**:
    - Reads and parses a text-based configuration file.
    - Supports nested structures using dot (.) notation for organizing options hierarchically.
    - Dynamically selects the appropriate processor type based on configuration options.
2. **Processor Abstraction**:
    - Uses an abstract base class (Processor) with multiple derived classes (TextProcessor, NumericProcessor, ImageProcessor, AudioProcessor).
    - Leverages polymorphism to allow dynamic runtime behavior based on configuration.
3. **Factory Design Pattern**:
    - Implements a factory class (ProcessorFactory) to instantiate processor objects based on configuration inputs.
4. **Template-Based Processing**:
    - Demonstrates the use of templates to process collections of data generically.
5. **Exception Handling**:
    - Provides robust error handling for invalid configurations, missing files, or unsupported processor types.

Learning Objectives

By completing this project, students will:

1. **Master Core C++ Concepts**:
    - Understand and implement polymorphism using abstract classes and inheritance.
    - Use STL containers like std::map and std::vector for efficient data management.
2. **Apply Design Patterns**:
    - Implement the Factory Design Pattern to decouple object creation from application logic.
3. **Handle File Operations**:
    - Read and parse text files dynamically to influence runtime behavior.
4. **Develop Robust Applications**:
    - Utilize exception handling to ensure the application remains stable under erroneous conditions.
5. **Explore Templates**:

- Create reusable, type-agnostic code for processing collections of data.

Project Workflow

1. **Configuration File**:
   - Prepare a configuration file (settings.txt) specifying the processor type and related options.
   - Example:

```
# Sample configuration for the Configurable Processor system.
# Set the processor type: Text, Numeric, Image, or Audio.
Processor.Options.Type=Image
Processor.Options.Threshold=0.5
Processor.Options.MaxRetries=3
```

2. **Dynamic Processor Creation**:
   - The system reads the configuration file at runtime and dynamically selects the appropriate processor class.
3. **Data Processing**:
   - Provide sample data to the system, and the chosen processor will process it based on its specialization.
4. **Robust Error Handling**:
   - Handle missing configuration files, invalid processor types, and unexpected runtime issues gracefully.

Example Execution

Configuration File:

```
# Sample configuration for the Configurable Processor system.
# Set the processor type: Text, Numeric, Image, or Audio.
Processor.Options.Type=Image
Processor.Options.Threshold=0.5
Processor.Options.MaxRetries=3
```

Output:

```
ImageProcessor processing: Sample Data (interpreted as image data)
```

Project Structure:

```
ConfigurableProcessor/
├── include/
│   ├── Processor.hpp
│   ├── ProcessorFactory.hpp
│   ├── Configuration.hpp
│   ├── Utilities.hpp
├── src/
│   ├── Processor.cpp
│   ├── ProcessorFactory.cpp
│   ├── Configuration.cpp
│   ├── main.cpp
├── config/
│   ├── settings.txt
├── Makefile
```

Makefile :

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall
SRCDIR = src
INCDIR = include
BUILDDIR = build
TARGET = main

SRC = $(wildcard $(SRCDIR)/*.cpp)
OBJ = $(patsubst $(SRCDIR)/%.cpp,
$(BUILDDIR)/%.o, $(SRC))

all: $(TARGET)

$(TARGET): $(OBJ)
$(CXX) $(CXXFLAGS) -o $@ $^

$(BUILDDIR)/%.o: $(SRCDIR)/%.cpp
@mkdir -p $(BUILDDIR)
$(CXX) $(CXXFLAGS) -I$(INCDIR) -c $< -o $@
```

Thank You
Edges For Training Team