GROUP 6: Lamport

Team Members:

Mehmet Efe Caylan
Emil Rahn-Siegel
Will Crawford
Ivan Bega
Orhan Aydin

ArgManager

Class Description:

A tool that takes in the “argc” and “argv” values supplied by the command lines and converts them into flags and settings. One option sets up the ArgManager to load in all of the command-line options before passing argc and argv. Can list all the options with “-help”, which can be shortened to “-h”, and more advanced options may also handle URL query params or interface with config files.

Similar Classes:

Std::Function
Std::Set

Key Functions:

LoadArgManager: Loads in all of the command line options
ConvertArg: takes argc and argv values and converts them into flags and settings
ListOptions: Lists all of the options

Error Conditions:

Programming errors:
Invalid argc
Invalid argv

User errors:
Invalid values
Invalid arguments provided

invalid URLs (If handling URL query)

Expected Challenges:

Making sure all of the commands have proper error checking. Dealing with invalid values, handling options. Implementing a more advanced version that handles URL query parameters or interfacing with config files may also pose a challenge. If there are many options implemented, implementing a function to list all of the options automatically may be needed.

Other Class Projects:

FunctionSet (Group 3)

CommandLine

Class Description:

A class to allow a user to specify their own command line inputs and what functions should be triggered when they input specific commands.

Similar Classes:

Std::Function

Key Functions:

GetCommands: the function to process the initial set of commands provided by the user
CallCommand: The function will receive a string representing a command from the user and then using a map it will match that command to a specific function and call that function

Error Conditions:

Any input provided by the user that isn't specified as part of the set of known commands. This will be a user error. Another error would be ensuring that calling functions connected to specific commands would always be done with a try-catch in case there is an issue with that function. This would be a programmer error.

Expected Challenges:

Any issues relating to interpreting the different commands. Making sure there is error checking in place for every input provided by the user. The other challenge would be learning how to use std::function.

Other Class Projects:

ActionMap

ErrorManager

Class Description:

This class will provide developers with tools for processing errors during program execution. Reasonable error messages (color coded) with descriptions, multiple error levels, ability to implement own functions that will be triggered in case of an error.

Similar Classes:

Std::function
std::exception
std::ofstream

Key Functions:

enum ErrorLevel
sendError(ErrorLevel, message) - Fancy error and warning output to the user
enableColor(bool)
assignColor(ErrorLevel, color)
setAction(ErrorLevel, Action) - custom action depending on error level
safeExecution(Function1, Function2) - execution with the possibility to recover - if a first function fails, call second function
enableProgramTermination(ErrorLevel, bool) - if an error of a certain type occurs - exit(1)
Setting up custom error message to be used for feedback for user input

Error Conditions:

Programmer errors:
references to invalid/null functions
Using invalid ANSI color codes when assigning color through a string
Trying to assign a nonexistent error level
User errors:
Invalid input that can’t be parsed
Recoverable errors:
Connection timeout when opening a web browser

Expected Challenges:

How do you determine if a function failed? Try-catch may be inefficient if we are expected to have frequent failures
Using lambda expression to pass functions

Other Class Projects:

FunctionSet (Group 3)
ActionMap (Group 2)

StaticString

Class Description:

A string type that enables users to define the static length of a string which provides a faster manipulation for strings without missing the base operators of a std::string type. This class aims to provide higher performance for string operations and manipulations.

Similar Classes:

Std::string
Std::array
Std::string_view
Std::span

Key Functions:

We will have the ability of;
comparing two strings (for both size and values),
creating a substring,
string concatenation,
Get size,
Get length,
copy or resize,
operator[],
find

Error Conditions:

User Errors:
Out of range: catching invalid attempts of accessing or modifying.
Invalid size: requesting invalid size for a substring.
Invalid argument (for comparison or copy): providing invalid strings to compare or copy to a smaller fixed-size string.
Doesn't Exist: trying to find a character doesn’t exist.

Programmer Error:
Undefined behavior: accessing a null ptr or trying to change.
Out of range: trying to access or modify the static string even if the value is valid, and in between static size, they might be empty.

Recoverable Error:
Invalid size: requesting invalid size for a copy or other operations.
Out of memory: requesting a size larger than memory, which would make the static string slower.

Expected Challenges:

We expect to face issues on out-of-index reaches or undefined behaviors for our class; since we have a static-sized array, we have to make sure we are on the same page all the time. We also need to make sure we do not have unnecessary copies whenever it is possible and should make sure not to leak memory due to manual allocation.

Other Class Projects:

Group 4, Audited String
Group 4, Assert

StringSet

Class Description:

Manage string collections based on StaticString class, which handles various set operations such as union, subtract, get strings with specific conditions, rules(maybe use regular expressions to define the rules), intersection, differences and similarities between two set or more.

A templated string set that lets a user choose either a static string or std::string for the set operations.

Similar Classes:

Std::set,unordered_set c++ classes similar to that.

Key Functions:

Union: Combine one or more(multiple) sets into a single set
Intersection: Find common elements between two or more sets
Difference: Find elements in a set that are not in another set
Filter: Find elements that satisfy specific conditions. For example, find strings that start with the character ‘o’.
Filter out: Remove elements that satisfy specific conditions.
Count: Count specific elements.
Iterator: Provide iterator functionality to our class.

Error Conditions:

Empty set: Perform operations on empty sets
Iterator: Dereferencing invalid iterator, out of range.

Expected Challenges:

Efficiency: Programmer error, using assert may be helpful
Unexpected behavior while handling large sets of strings: Potentially recoverable error, using exceptions, providing reasonable error messages may be helpful.
Managing invalid inputs: User error, we can use exceptions.
Working with nested sets: Programmer error, we can use assert.

Other Class Projects:

Project 4, Index Set