command-line-parser

João Gaspar Oliveira Cunha

April 7, 2023

Version 1.0

Contents

List of Figures

1 Introduction

Disclaimer: *This project is a C++ copy, of the Case study of a command-line argument parser as seen in Clean Code Chapter 14 - Successive refinement.*

1.1 *Raison d'être*

The main *reason for being* of this framework was to try and implement the command line argument parser in C++ as an exercise. As suggested in the book.

1.2 How to use

Let's go through the steps:

1. Include the necessary header files:

```
#include "args.h"
#include "argsException.h"
#include <iostream>
#include <vector>
```

   The `args.h` header file contains the declaration of the `Args` class, and the `argsException.h` header file contains the declaration of the `ArgsException` class.

2. Create an instance of `Args` class:

```
Args arg("l,p#,d*,", args);
```

   Here, an instance of the `Args` class is created with the argument format string `"l,p#,d*,"` and the command-line arguments passed to the program are stored in a vector called `args`.

3. Retrieve values for different types of arguments:

```
int integer = arg.getInt('p');
bool boolean = arg.getBoolean('l');
string string = arg.getString('d');
```

   Using the instance of `Args` class, you can retrieve values for different types of arguments. In this example, `getInt()`, `getBoolean()`, and `getString()` member functions of the `Args` class are called with the corresponding argument names `('p', 'l', 'd')` to retrieve `integer`, `boolean`, and `string` values respectively.

4. Handle exceptions:

```
catch (ArgsException& e)
{
    std::cerr << "Argument␣error:␣" + e.getErrorMessage() + "\n";
}
```

   If any exception occurs while parsing the command-line arguments using the `Args` class, an `ArgsException` object is thrown. You can catch this exception and handle the error by calling the `getErrorMessage()` member function of the `ArgsException` class to retrieve the error message, which can then be printed to the standard error stream `(std::cerr)` for error reporting.

5. Use the retrieved values as needed:

```
    catch (ArgsException& e)
    cout << "Integer␣inserted␣was:␣" << integer << endl;
    cout << "Boolean␣inserted␣was:␣" << boolean << endl;
    cout << "String␣inserted␣was:␣" << string << endl;
```

After retrieving the values for different arguments using the `Args` class, you can use them as needed in your program. In this example, the retrieved values are printed to the standard output `(std::cout)` for display.

Note: The argument format string `"l,p#,d*,"` used in the example specifies the expected format of the command-line arguments. The characters `'l'`, `'p'`, and `'d'` represent the argument names, and the special characters `'#'` and `'*'` indicate that the corresponding arguments should be followed by an integer and a string respectively. Please refer to the documentation of the `Args` class for more details on how to specify argument formats.

2 Classes

2.1 The `Args` class

The `Args` class is a C++ implementation of a command line argument parser. It is designed to parse command line arguments based on a given schema and a list of argument strings.

Members:

- `marshalers`: An unordered map that stores argument marshallers, where the keys are characters representing the argument IDs, and the values are shared pointers to instances of subclasses of the `ArgumentMarshaller` class, which is an abstract base class for argument marshallers.
- `argsFound`: A set that keeps track of the arguments found during parsing.
- `currentArgument`: An iterator pointing to the current argument being parsed.

Methods:

- `Args(std::string schema, std::vectorstd::string args)`: Constructor that takes a schema string and a vector of argument strings as input. It initializes the member variables and calls the `parseSchema()` and `parseArgumentStrings()` methods to parse the schema and argument strings, respectively.
- `parseSchema(std::string& schema)`: Private method that parses the schema string and populates the `marshalers` map with argument marshallers based on the schema elements.
- `parseSchemaElement(const std::string& element)`: Private method that parses an individual schema element and creates an argument marshaller based on the element's ID and tail.
- `validateSchemaElementId(char elementId)`: Private method that validates if the given schema element ID is a valid character (alphabetical).
- `parseArgumentStrings(std::vectorstd::string& args)`: Private method that iterates through the argument strings and calls the `parseArgumentCharacters()` method to parse each argument string.
- `parseArgumentCharacters(const std::string& argChars)`: Private method that iterates through the characters of an argument string and calls the `parseArgumentCharacter()` method to parse each character.
- `parseArgumentCharacter(char argChar)`: Private method that parses an individual argument character, finds the corresponding argument marshaller from the `marshalers` map, and sets the argument value using the marshaller's `set()` method.
- `has(char arg)`: Public method that checks if the given argument ID is found in the `argsFound` set, indicating whether the argument was parsed.
- `getBoolean(char arg)`: Public method that retrieves the boolean value of the argument with the given ID using the corresponding boolean argument marshaller's `getValue()` method.

- `getString(char arg)`: Public method that retrieves the string value of the argument with the given ID using the corresponding string argument marshaller's `getValue()` method.
- `getInt(char arg)`: Public method that retrieves the integer value of the argument with the given ID using the corresponding integer argument marshaller's `getValue()` method. If the argument is not found or not of type `IntegerArgumentMarshaller`, it returns 0.

2.2 The `ArgumentMarshaller` class

The `ArgumentMarshaller` class is an abstract base class for argument marshallers in a C++ implementation of a command line argument parser. It provides a pure virtual method `set()` that takes an iterator pointing to the current argument being parsed and is meant to be overridden by subclasses.

Methods:

- `set(std::vectorstd::string::iterator currentArgument)`: A pure virtual method that sets the argument value based on the current argument being parsed. Subclasses of `ArgumentMarshaller` are expected to implement this method according to their specific argument types.

2.3 The `BooleanArgumentMarshaller` class

The `BooleanArgumentMarshaller` class is a subclass of `ArgumentMarshaller` in a C++ implementation of a command line argument parser. It is responsible for marshalling boolean arguments. It has a boolean member variable `booleanValue` to store the boolean value of the argument.

Members:

- `booleanValue`: A boolean member variable that stores the boolean value of the argument.

Methods:

- `BooleanArgumentMarshaller()`: Constructor that initializes the `booleanValue` to false.
- `void set(std::vectorstd::string::iterator currentArgument)`: Overrides the `set()` method from the base class `ArgumentMarshaller`. It sets the `booleanValue` to true.
- `bool getValue()`: Method that retrieves the boolean value of the argument by returning the value of `booleanValue`.

2.4 The `IntegerArgumentMarshaller` class

The `IntegerArgumentMarshaller` class is a subclass of `ArgumentMarshaller` in a C++ implementation of a command line argument parser. It is responsible for marshalling integer arguments. It has an integer member variable `integerValue` to store the integer value of the argument.

Members:

- `integerValue`: An integer member variable that stores the integer value of the argument.

Methods:

- `IntegerArgumentMarshaller()`: Constructor that initializes the `integerValue` to 0.
- `void set(std::vectorstd::string::iterator currentArgument)`: Overrides the `set()` method from the base class `ArgumentMarshaller`. It attempts to convert the next element pointed by `currentArgument` to an integer using `stoi()` function, and stores the result in `integerValue`. If the conversion fails, it throws an `ArgsException` with the error code `MISSING_INTEGER`.
- `int getValue()`: Method that retrieves the integer value of the argument by returning the value of `integerValue`.

2.5 The `StringArgumentMarshaller` class

The `StringArgumentMarshaller` class is a subclass of `ArgumentMarshaller` in a C++ implementation of a command line argument parser. It is responsible for marshalling string arguments. It has a string member variable `stringValue` to store the string value of the argument.

Members:

- `stringValue`: A string member variable that stores the string value of the argument.

Methods:

- `StringArgumentMarshaller()`: Constructor that initializes the `stringValue` to an empty string.
- `void set(std::vectorstd::string::iterator currentArgument)`: Overrides the `set()` method from the base class `ArgumentMarshaller`. It sets the `stringValue` to the next element pointed by `currentArgument`. If the element is missing, it throws an `ArgsException` with the error code `MISSING_STRING`.
- `std::string getValue()`: Method that retrieves the string value of the argument by returning the value of `stringValue`.

2.6 The `ArgsException` class

This class represents an exception that can be thrown in case of errors related to command-line arguments parsing. It is derived from the standard `std::exception` class.

Members:

- `char errorArgumentId`: A character representing the error argument ID.
- `std::string errorParameter`: A string representing the error parameter.
- `ErrorCode errorCode`: An enumeration representing the error code.

- `std::string errorMessage`: A string representing the error message.

Methods:

- `ArgsException()`: Default constructor that initializes the members with default values.
- `ArgsException(const std::string& message)`: Constructor that takes a string parameter representing the error message and sets it as the error message.
- `ArgsException(const ErrorCode errorCode)`: Constructor that takes an `ErrorCode` parameter and sets it as the error code. It also sets the error message based on the error code.
- `ArgsException(const ErrorCode errorCode, const std::string& errorParameter)`: Constructor that takes an `ErrorCode` parameter and a string parameter representing the error parameter, and sets them as the error code and error parameter respectively. It also sets the error message based on the error code and error parameter.
- `ArgsException(const ErrorCode errorCode, const char errorArgumentId, const std::string& errorParam` Constructor that takes an `ErrorCode` parameter, a character representing the error argument ID, and an optional string parameter representing the error parameter. It sets them as the error code, error argument ID, and error parameter respectively. It also sets the error message based on the error code, error argument ID, and error parameter.
- `const char* what() const noexcept`: Overrides the `what()` method from the base `std::exception` class. It returns a pointer to the error message string.
- `char getErrorArgumentId()`: Returns the error argument ID.
- `void setErrorArgumentId(const char errorArgumentId)`: Sets the error argument ID.
- `std::string getErrorParameter()`: Returns the error parameter.
- `void setErrorParameter(const std::string& errorParameter)`: Sets the error parameter.
- `ErrorCode getErrorCode()`: Returns the error code.
- `void setErrorCode(ErrorCode errorCode)`: Sets the error code.
- `std::string getErrorMessage()`: Returns the error message based on the error code, error argument ID, and error parameter.

Generated by ChatGPT.