МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ «МИСИС»

Институт информационных технологий и компьютерных наук

Кафедра инженерной кибернетики

Курсовая работа

по дисциплине

«Объектно-ориентированное программирование»

на тему

«Парсер аргументов командной строки»

Выполнил:
студентка 1-го курса,
гр. БПМ-22-2 Карнаух М.В.

Проверил:
доцент, к.т.н. Полевой Д.В.

Москва, 2023

Оглавление

Описание задачи

Необходимо реализовать библиотеку argParser, предназначенную для упрощения обработки и анализа аргументов командной строки. Библиотека argParser предназначена для разработчиков программ и приложений, которым требуется обработка аргументов командной строки. Библиотека позволяет легко определить и добавить различные типы аргументов (строки, целые числа, логические значения) с поддержкой значений по умолчанию, описаний и других параметров. Библиотека обеспечивает удобный и гибкий способ разбора аргументов командной строки и доступа к их значениям в программе.

Библиотека предоставляет следующие возможности:

1. Возможность добавления различных типов аргументов (строки, целые числа, логические значения).
2. Поддержка коротких и длинных имен аргументов (с префиксами "-" и "--").
3. Возможность задания значений по умолчанию для аргументов.
4. Возможность описания аргументов для вывода справки и подсказок пользователю.
5. Поддержка аргументов с несколькими значениями (множественные аргументы).
6. Разбор аргументов командной строки и доступ к их значениям в программе.
7. Проверка правильности типов и значений аргументов, вывод сообщений об ошибках при несоответствии.

Пользовательское описание

Библиотека argParser предоставляет удобный способ разбора аргументов командной строки в программных приложениях. С помощью этой библиотеки можно определить и обрабатывать различные типы аргументов, такие как строки, целые числа и флаги (логические значения), передаваемые через командную строку.

Для начала работы с библиотекой, необходимо установить исходники (раздел «Установка и сборка библиотеки»), подключить её к вашему проекту (`#include <argParser/argParser.hpp>`) и использовать пространство имён `ArgumentParser`.

Основные шаги для работы с библиотекой argParser:

1. Создание парсера аргументов:
   Создайте объект класса `ArgParser`, указав имя вашей программы в качестве параметра конструктора.
   Например: `ArgParser parser("MyParser");`

2. Определение аргументов:
   Добавьте аргументы с помощью методов `AddStringArgument`, `AddIntArgument` или `AddFlag`, в зависимости от типа аргумента. Установите имя аргумента, короткое имя (если требуется).
   Например: `parser.AddIntArgument('i', "input");`

3. Методы `MultiValue`, `Default` и `Description` позволяют определить аргументы с множественными значениями, установить значение/значения по умолчанию, и добавить описание для аргумента.
   Например: `parser.AddIntArgument('i', "input").MultiValue(3);`

4. Определение позиционных аргументов:
   Методы `AddIntPositional` и `AddStringPositional` позволяют добавлять позиционные аргументы целочисленного и строкового типов.
   Метод `Store` позволяет сохранять значения позиционных аргументов в векторе программы для дальнейшего использования.
   Например: `parser.AddIntPositional(2).Store(values);`

5. Обработка аргументов:
   Вызовите метод `Parse`, передав в него аргументы командной строки, полученные при запуске программы.
   Например: `parser.Parse(argc, argv);`

   Метод `Parse` проанализирует переданные аргументы и заполнит значения аргументов, а также выполнит необходимые проверки.

6. Извлечение значений аргументов:
   Используйте методы `GetStringValue`, `GetIntValue` и `GetFlag` для извлечения значений аргументов по их имени.
   Если аргумент имеет множественные значения, вы можете указать индекс для извлечения конкретного значения.

   Используйте методы `GetStringPositional`, `GetIntPositional` для извлечения значений позиционных аргументов по их индексу.

   Например: `parser.GetIntValue("input", *индекс*);`

7. Дополнительные возможности:
   Метод `StoreValue` и `StoreValues` (для multi-value аргументов) позволяют сохранять значения аргументов в переменных программы для дальнейшего использования.
   Например: `parser.AddIntArgument('i', "input").StoreValue(val);`

   Методы `GetPositionalCount` и `GetArgumentCount` позволяют получить количество позиционных аргументов и количество значений аргумента по его имени.

   Например: `parser.GetArgumentCount("input");`

   При передаче аргумента `--help` выводится справочное сообщение, которое содержит описание всех доступных аргументов.
   Например: `./myProgram --help`

8. Запуск программы:
   При запуске программы через командную строку передайте аргументы и их значения. Если есть позиционные аргументы, то передайте их в начале, далее передавайте аргументы в виде: "-*короткое имя*" или "--*длинное имя*". После каждого аргумента вводятся его значения (значений может и не быть).
   Например: `./myProgram first.txt second.txt --input 1 2 3;`
   (first.txt и second.txt являются позиционными аргументами, --input – аргумент, 1 2 3 – значения аргумента input)

Техническое описание

Структуры и классы:

`ArgType` - enum класс для перечисления возможных типов аргументов.

`Argument` - шаблонный класс для хранения свойств аргумента.

`ArgParser` - основной класс для парсинга аргументов.

Документация по классам, сгенерированная с помощью DoxyGen:

Класс ArgumentParser::ArgParser

Класс, представляющий анализатор аргументов командной строки.
`#include <argParser.hpp>`

Открытые члены

`ArgParser` `(const std::string &name)`
*Создает объект **ArgParser** с заданным именем программы.*

`Argument``< std::string > &` **`AddStringArgument`** `(const std::string &name)`
Добавляет строковый аргумент с указанным именем.

`Argument``< std::string > &` **`AddStringArgument`** `(char shortName, const std::string &name)`
Добавляет строковый аргумент с указанным коротким именем и полным именем.

`Argument``< int > &` **`AddIntArgument`** `(const std::string &name)`
Добавляет целочисленный аргумент с указанным именем.

`Argument``< int > &` **`AddIntArgument`** `(char shortName, const std::string &name)`
Добавляет целочисленный аргумент с указанным коротким именем и полным именем.

Argument< bool > & **AddFlag** (char shortName, const std::string &name)
Добавляет аргумент-флаг с указанным коротким именем и полным именем.

bool **Parse** (std::vector< std::string > &args)
Анализирует аргументы командной строки и заполняет значения аргументов.

bool **Parse** (int argc, char **argv)
Анализирует аргументы командной строки и заполняет значения аргументов.

void **PrintHelp** () const
Выводит справочное сообщение со списком доступных аргументов.

ArgParser & **AddIntPositional** (size_t posCount=1)
Добавляет целочисленные позиционные аргументы.

ArgParser & **AddStringPositional** (size_t posCount=1)
Добавляет строковые позиционные аргументы.

std::string **GetStringValue** (const std::string &name, const int index=0) const
Извлекает строковое значение указанного аргумента.

int **GetIntValue** (const std::string &name, const int index=0) const
Извлекает целочисленное значение указанного аргумента.

bool **GetFlag** (const std::string &name) const
Извлекает логическое значение указанного аргумента.

std::string **GetStringPositional** (const int index) const
Извлекает строковый позиционный аргумент указанного индекса.

int **GetIntPositional** (const int index) const
Извлекает целочисленный позиционный аргумент указанного индекса.

size_t **GetPositionalCount** () const
Возвращает количество позиционных аргументов.

size_t **GetArgumentCount** (const std::string &name) const
Возвращает количество значений аргумента.

Шаблон класса ArgumentParser::Argument< T >

Класс, представляющая аргумент.
`#include <argParser.hpp>`

Открытые члены

Argument< T > & **Default** (const T &val)
Устанавливает значение по умолчанию для ранее добавленного аргумента.

Argument< T > & **Default** (const std::vector< T > &val)
Устанавливает значения по умолчанию для ранее добавленного аргумента.

Argument< T > & **MultiValue** (size_t minArgsCounter=1)
Задает минимальное количество аргументов для ранее добавленного мульти-аргумента.

Argument< T > & **StoreValue** (T &variable)
Сохраняет значение ранее добавленного аргумента в указанной переменной.

Argument< T > & **StoreValues** (std::vector< T > &variable)
Сохраняет значения ранее добавленного аргумента в указанной переменной.

Argument< T > & **AddDescription** (const std::string &descr)
Добавляет описание ранее добавленного аргумента.

void **SetValue** (const T &**value**)
Задает значение добавленного аргумента.

Установка и сборка библиотеки

Файлы и документация расположены по ссылке на репозиторий github [1]. Оттуда необходимо клонировать репозиторий командой

```
git clone https://github.com/marieKarnaukh/karnaukh_m_v.git <имя папки>
```

После этого необходимо собрать проект с помощью CMake (версия должна быть 3.18 или выше). Сборка осуществляется вручную, следуя данному алгоритму:

1. Перейдите в директорию, в которую установили репозиторий (<имя папки>, по-умолчанию - karnaukh_m_v). Далее перейдите в директорию karnaukh_m_v.
2. Откройте командную строку (GitBash для Windows, Shell для Linux) и напишите команду “`cmake -S . -B build -DCMAKE_TOOLCHAIN_FILE=<путь до vcpkg.cmake> && cmake --build build --config Release`”. Для инсталяции испольняемого файла напишите команду “`cmake --install build --component parser --prefix <путь инсталляции>`” (`--component parser_lib` для инсталяции библиотеки).
3. Документация будет сгенерирована в директорию docs.

Тестирование

После сборки вы можете протестировать пример использования библиотеки. Для этого вам нужно будет выполнить следующие действия из командной строки:

1. Перейти в папку инсталляции. Затем перейдите в папку bin.
2. Создайте и откройте файл input.txt, запишите в него какой-либо текст.

   Для открытия файла и вывода его содержимого, передайте имя файла в качестве позиционного аргумента, а также опцию --print или -p.

   ```
   ./parser input.txt --print
   ```

   Программа выведет содержимое файла input.txt на экран. Убедитесь, что выводится то, что вы записали в файл ранее.
3. Для записи значения в файл, передайте имя файла в качестве позиционного аргумента, а также опцию --write или -w, за которой следует значение для записи.

   ```
   ./parser output.txt --write Hello, world!
   ```

   Программа создаст или перезапишет файл output.txt и запишет в него строку "Hello, world!".
4. Убедитесь в корректности записи, для этого выведите его содержимое с помощью опции --print или -p.

   ```
   ./parser output.txt --print
   ```
5. Можно комбинировать опции --print и --write для одного запуска программы.

   ```
   ./parser input.txt --write Hello, argParser! --print
   ```

   Программа сначала создаст или перезапишет файл input.txt и запишет в него строку "Hello, argParser!", а затем выведет содержимое файла input.txt.

   Убедитесь, что в качестве содержимого input.txt выводится "Hello, argParser!", которое было записано с помощью --write.
6. Проверьте, как программа обрабатывает некорректные или отсутствующие аргументы. Удостоверьтесь, что программа выводит соответствующие сообщения об ошибке, когда файл не может быть открыт или не указано значение для записи.

   Попробуйте открыть несуществующий файл:

   ```
   ./parser file1.txt --print
   ```

   Убедитесь, что получили ошибку «Failed to open file: file1.txt».

   Попробуйте не указывать значение для аргумента --write:

   ```
   ./parser input.txt --write
   ```

   Убедитесь, что получили ошибку «Not enough values for 'write' argument».
7. Проверьте вывод справочного сообщения с помощью опции --help:

   ```
   ./parser --help
   ```

   Убедитесь, что получили информацию об аргументах --print(-p) и --write(-w).

Список использованной литературы

1 Репозиторий GitHub - URL: https://github.com/marieKarnaukh/karnaukh_m_v (дата обращения 05.06.2023).