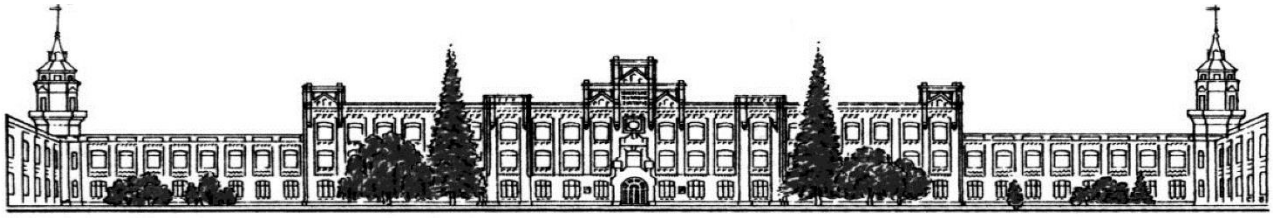

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА №2

З ДИСЦИПЛІНИ "Технології програмування для комп'ютерних систем-3"

Виконав:
студент групи ІВ-91мн
Гончаренко О.О.
Перевірив:
Регіда П.Г.

м. Київ – 2020 р.

Завдання

Написати програму на C++, яка приймає і аналізує будь-яку кількість параметрів командного рядка.

(Поправка: відповідна програма була написана на Java. В принципі, можна було використати конвертор, але, враховуючи складну ієрархію класів, це була б погана ідея)

Вимоги до програми

Обробка довгих та коротких форм ключа.

Повторення ключів ігноруються (ключ передається двічі або в різних формах).

Виведення відповідного повідомлення для невідомого параметра.

Після розбору повинні бути надруковані тільки унікальні аргументи.

Правила задання та роботи команд

Команда може містити наступні елементи:

- Короткі ключі:
    - починаються символом «-».
    - Аргументи для ключа слідують після ключа, через пробіл, розділяються пробілами
    - Аргументів має бути рівно стільки, скільки вказано в описанні ключа
    - Короткі ключі без аргументів можуть вказуватись разом:
        - *Cmd -abcde*
    - Короткі ключі з аргументами мають вказуватись окремо.
        - Cmd -a 2 -b 5 -c hello
    - Короткі ключі завжди складаються лише з одного символу
        - -ab – два різні ключі, але ні в якому разі не 1 короткий ключ
- Довгі ключі
    - Починаються символами «--».
    - Аргументи слідують після ключа без пробілу, через символ «=», розділяються комами. Ні в якому разі аргументи не мають вказуватись через пробіл або містити пробілів!
    - Довгий ключ дозволяє змінне число аргументів (тобто, за специфікацією аргументів 2, а по факту може бути 1 чи 3, і це не буде вважатись помилкою). Обробка таких випадків повністю покладена на

обробник ключа (тобто, якщо така поведінка була врахована – все буде добре. Якщо ні – буде виключення і переривання команди).

- Параметри. Параметром може бути навіть ім'я іншої команди (в системі команд є команди, що викликають інші команди). Параметри ЗАВЖДИ ідуть після ключів. Якщо спочатку вказати параметр, а потім – ключ, то цей ключ не буде сприйнятий системою як ключ, а буде розглядатись як параметр.
- Параметром може бути підкоманда
- Всі ключі, що йдуть після імені підкоманди, сприймаються як дані для підкоманди .

Архітектура програми

Поділяється на такі частини:

1. Рівень CLI

Містить клас CLI та його реалізації. Основний метод CLI – *start()*. Його алгоритм наступний:

1. На потік вводу виводиться запрошення
2. З потоку вводу читається команда
3. Вхідний рядок розбивається на аргументи (*String[] args*)
4. Ці аргументи передаються в основну команду (*main command*). Ця команда завжди є в системі.

2. Рівень команд

На цьому рівні відбувається обробка команд. Основний метод – *main()*. Він приймає на вхід вхідний масив (*String[] args*), поточне зміщення (*int offset*) та потоки вводу-виводу (*IOE*), через які команда має взаємодіяти з користувачем.

Є 2 типи команд: прості команди (не мають ключів, все що йде після імені команди сприймається як аргументи) та складні команди (мають ключі, можуть їх відрізняти від аргументів).

Прості команди реалізовано через наслідування від *AbstractCommand*. Є кілька команд-шаблонів. Найцікавіший – клас *SimpleChainCommand*. Цікавий він тим, що ним реалізовується основна команда. Він містить пул команд і обробник «за замовчуванням». Коли користувач вводить команду, аналізується перший елемент масиву *args* (у випадку основної команди, а взагалі *SimpleChainCommand* може

використовуватись і для інших цілей) і шукається відповідність в пулі. Якщо є – викликається відповідний обробник. Цей обробник – теж команда. Його методу *main()* подаються ті ж самі аргументи, але зміщення змінюється на *offset+1*. Це дозволяє робити ланцюги команд і навіть рекурсивні виклики за необхідності.

Складні команди працюють в 3 етапи: парсинг, попередня обробка і, власне, виконання.

3. Парсинг складних команд

Ідея наступна: є база даних ключів та парсер. В базі зберігаються всі ключі конкретної команди. Для кожного ключа зберігаються *унікальний ID*, *довга форма* (обов'язково), *коротка форма* (необов'язково) та *число аргументів* (за замовчуванням 0). База нічого не знає про інші елементи програми, лише надає доступ до даних. Вона ініціалізується на початку роботи програми. Для бази з ідентифікацією по довгому ключу зроблено наступне: вона може бути проініціалізована рядком. Формат рядка можна описати так:

<string>::= <entry>|<entry>\n<entry>\n…<entry>

<entry>::= <long key> [<short key>] [<argument count>]

<long key>::=<string>

<short key>::=<char>|-

<argument count>::=<int>|-

Приклад:

Рядок (файл)	Довгий ключ (primary key)	Короткий ключ	Число аргументів
test t 0	test	t	0
version v 1	version	v	1
help h	help	h	0
no_short_key – 1	no_short_key	<null>	1
noshortnargs	noshortnargs	<null>	0

Парсер аналізує кожен аргумент і формує запити до бази. На виході формується хеш-мапа з опціями та аргументами до них (за збереження аргументів відповідає інтерфейс *Option*, всі аргументи зберігаються як рядки, всі проблеми із семантикою покладаються на обробник опції) та зміщення початку аргументів.

4. Попередня обробка

Цей етап необхідний для таких опцій як help чи інші тому подібні. В першу чергу це виконання додаткових дій, які не залежать від самої команди. За це відповідає інтерфейс *OptionHandler*. Він містить єдиний метод *handle()*.

Також кожна команда має окрему базу даних для попередніх обробників. Ця база незалежна від бази, що описує ключі. На жаль, обробники текстом описувати проблематично, тому ініціалізація бази має виконуватись через написання відповідного коду.

На щастя, є кілька шаблонів обробників опцій.

- *Return* – повертає код і більше ніяких дій не виконує (службовий шаблон)
- *Help* – виводить в потік виводу заздалегідь заданий текст
- *Echo* – виводить в потік виводу всі аргументи опції (корисно для перевірки)
- *Stacked* – досить цікавий шаблон. Запускає кілька інших обробників в заданому порядку з одними і тими ж аргументами. Корисний якщо потрібно виконати кілька шаблонних дій послідовно і не хочеться писати для цього код.
- *Option as Command* – ще більш цікавий шаблон. Дозволяє викликати просту команду, передавши в якості аргументів аргументи ключа. Хто сказав що лише команда може викликати команду?!

5. Виконання команди

Цим займаються клас *CommandHandler*. Його основний і єдиний метод – *execute()*. Обробник у команди лише один. Також на даний момент створено кілька шаблонів. Один із них – *Command as Option*. І він робить те ж, що і *Option as Command*, але навпаки: перетворює аргументи команди в аргументи опції. Це дозволяє використовувати вже наявні обробники опцій коли це потрібно.

Опис прикладу

Для тесту створено TestCLI, що має наступну систему команд:

Команда	Опис
nop	Нічого не робить
exit	Завершує програму

help <command>	Виводить довідку.
test [keys] [arguments]	Тест роботи програми. Має ключі. Докладний опис – в довідці.

Робота програми

```
Test CLI > help
this command prints manuals. There are next command you can use:
* nop
* exit
* help
* test

Test CLI > help test
Prints arguments to console (echo). Keys:
    --test (-t): starts 3 handlers, print feedback to out
    --help (-h): print this message
    --number (-n): print number
    --array (-a): print 2 numbers or more (similar to -n)
    --silence (-s): blocks command's echo
    --exit (-e): quits from CLI after command's finishing

Test CLI > test -a 1 2 3
array : [1, 2]
test : [3]
Test CLI > test --array=1,2,3 4.9
array : [1, 2, 3]
test : [4.9]
Test CLI > test
test : []
Test CLI > test -s 1 2 3
Test CLI > test -s --silence 1 2 3
[WARNING] Arg # 2: duplicated option detected: silence, token: --silence for command test
[WARNING] Arg # 2: duplicated option detected: silence, token: --silence for command test
Solution: rewrite option state
Solution: rewrite option state
Test CLI > test -hs
Prints arguments to console (echo). Keys:
    --test (-t): starts 3 handlers, print feedback to out
    --help (-h): print this message
    --number (-n): print number
    --array (-a): print 2 numbers or more (similar to -n)
    --silence (-s): blocks command's echo
    --exit (-e): quits from CLI after command's finishing
Test CLI > test --test -t 1 2 3 4 5
[WARNING] Arg # 2: duplicated option detected: test, token: -t for command test
[WARNING] Arg # 2: duplicated option detected: test, token: -t for command test
Solution: rewrite option state
First handler works!
Solution: rewrite option state
Second handler works!
Third handler works!
test : [1, 2, 3, 4, 5]
Test CLI > test --number=1,2 -se this_message_will_not_be_printed! --help
number : [1, 2]

Process finished with exit code 0
```

Коментарі:

1. *help*

Команда вивела список команд

2. *help test*

Команда вивела довідку по команді “test”

3. *test -a 1 2 3*

Вивід масиву. Тут аргументами ключа є лише аргументи 1 і 2 (бо ключ -a має 2 аргументи). Аргумент 3 відноситься до програми. Чому це так? Див. «Правила задання та роботи команд»

4. *test --array=1,2,3 4.9*

Вивід 1,2 і 3 як аргументів ключа. І вивід 4.9 як аргументу команди. Цей приклад наочно іллюструє різницю роботи коротких та довгих ключів.

5. *test*

Просто вивід команди без аргументів. Як видно, виведено лише назву команду і порожні квадратні дужки (порожній масив аргументів).

6. *test -s 1 2 3*

Команда має 3 аргументи. Але ключ -s блокує вивід. І, як наслідок, виводу немає.

7. *test -s --silence 1 2 3*

Ключ -s – скорочена форма “silence”. Бачимо повідомлення про помилку. Дублювання помилки це нормально: вивід йде в потік помилки (System.err) і в потік для логу (замість нього зараз встановлено звичайний System.out). Звідси і дублювання повідомлень.

8. *test -hs*

Увімкнено ключі h (вивід довідки) та s (блокування виводу команди). Здавалося б , вивід заблоковано? А ні, не заблоковано, бо довідка виводиться ключем. Просто ще один цікавий приклад «на уважність» .

9. *test -t --test 1 2 3 4 5*

Повідомлення про помилку є. Також виведено повідомлення про виконання 3 обробників (тест того, як працює ланцюгова обробка). І, в кінці, очікуваний масив.

10. *test –number=1,2 -se this_message_will_not_be_printed! –help*

Останній приклад. Вже було сказано, що довгий ключ може приймати не ту кількість аргументів, що вимагається. Number виводить 1 число, але тут для цього

ключа задана також поведінка на випадок наявності кількох аргументів. І тому помилки не виникає. Також встановлено прапори -s та -e: вимикання виводу та завершення програми. Після цього йде повідомлення, що не виводиться (зрозуміло чому) і ключ --help. Програма не сприймає його як ключ: він слідує після аргументу. Але й помилки немає і це нормально: оскільки програмою підтримується субкомандинг (тобто, виклик командою іншої команди) то випадки, коли йде команда, ключі, параметри, потім підкоманда і ключі вже для неї це нормально. Але якби це була, наприклад, команда для складання чисел з аргументами-числами і серед параметрів стояв би ключ то це призвело б до завершення команди через семантичну помилку.

Висновки

Було розроблено консольний інтерфейс користувача (CLI). Ключовою перевагою даного рішення є те, що описана програма не просто один-єдиний CLI, а конструктор для побудови CLI. Будь-хто може підключити дане рішення як бібліотеку в свій проект (файл .jar присутній в out/artifacts/CLI_jar), реалізувати кілька своїх класів з потрібною логікою, описати поведінку команд (для спрощення цього і придумані шаблони), все це закинути до консольного інтерфейсу і використовувати його для своєї програми.

Недоліком є час розробки. Як виявилось, це не те що не просто, а зовсім не просто. Треба було писати просту програму на C++, а не це жахіття. Інший недолік це недостатньо велика кількість шаблонів та все ще складна робота з опціями. Чому це так – див. перший недолік.