Практика 1: задача про CLI

07.02.2022

1. Задача про CLI

Начнём мы с задачи на попроектировать, пока как умеете. Задача такая: спроектировать простой интерпретатор командной строки, поддерживающий следующие команды:

- cat [FILE] — вывести на экран содержимое файла;
- echo — вывести на экран свой аргумент (или аргументы);
- wc [FILE] — вывести количество строк, слов и байт в файле;
- pwd — распечатать текущую директорию;
- exit — выйти из интерпретатора.

Кроме того, должны поддерживаться одинарные и двойные кавычки (full and weak quoting, то есть одинарные кавычки передают текст как есть, двойные выполняют подстановки переменных окружения с оператором $), собственно окружение (команды вида «имя=значение»), оператор $, вызов внешней программы через Process (или его аналоги) для любой команды, которую интерпретатор не знает. Должны ещё поддерживаться пайплайны (оператор «|»), то есть перенаправление ввода и вывода. Примеры:

```
> echo "Hello, world!"
Hello, world!

> FILE=example.txt
> cat $FILE
Some example text

> cat example.txt | wc
1 3 18

> echo 123 | wc
1 1 3

> x=ex
> y=it
> $x$y
```

Проектировать систему надо так, чтобы новые команды было добавлять легко (логично, что шелл будет постепенно расширяться новыми встроенными командами), но желательно поддерживать архитектуру достаточно простой и слабо связанной, чтобы можно было реализовать и другие требования, которые могут возникать по ходу. Может потребоваться внезапно реализовать ещё что-нибудь из того, что умеют обычные шеллы, как и в реальной жизни, желания заказчика непредсказуемы (поэтому, кстати, не надо пытаться их предугадать и заложить в архитектуру — то, о чём вы подумали, никогда не случится, случится то, о чём вы не подумали).

Собственно эта задача станет и домашней, и начать её делать надо в аудитории. Итоговым результатом должен стать документ, описывающий основные архитектурные решения, достаточно подробно, чтобы в процессе кодирования не надо было задумываться и о чём «архитектурном». Должна быть структурная диаграмма, как умеете: умеете UML — рисуйте диаграмму классов, не умеете — сойдёт структурная схема из соединённых стрелочками прямоугольников, лишь бы по ней было понятно, где что, и что примерно делает. Должно быть также и текстовое описание, поясняющее происходящее на диаграмме (примерно две-три страницы текста, описывающие каждую сущность на диаграмме и как они взаимодействуют). Обратите внимание, что код писать пока не надо — через неделю будет отдельное задание это реализовать (потом при проверке мы посмотрим, насколько реализация соответствует архитектуре).

На паре надо, во-первых, поделиться на команды по два-три человека. Во-вторых, проанализировать условие, выявить неоднозначности и вообще места, где условие недостаточно подробно. Может быть, посмотреть, как работают реальные шеллы. Дальше надо выполнить декомпозицию системы на компоненты, классы и основные методы, нарисовать первое приближение диаграммы (только не увлекайтесь, сделать её достаточно детальной даже в команде из трёх человек за пару не успеть), быть готовыми в конце пары рассказать решение. Текстовое описание пока не надо. В процессе анализа задавайте вопросы по условию, оно намеренно неподробно. Надо постараться за пару сделать так, чтобы все примерно представляли, как стали бы это писать (потому что потом реально надо будет это писать). Кстати, в какой-то момент в следующих домашках надо будет что-то реализовать в чужой архитектуре (не вашей команды), поэтому постарайтесь не халявить при проектировании.

Дома надо будет по результатам обсуждения уточнить архитектуру, дорисовать диаграмму и написать текстовое описание. На что обратить внимание:

- выполняйте проектирование сверху вниз — сначала определитесь с общей структурой системы, определитесь с компонентами, их ответственностью и связями между ними, и только после этого переходите к проектированию компонентов;
- не закапывайтесь в деталях — задача намеренно такая, что можно всю пару обсуждать только вопросы парсинга и подстановки или особенности поведения wc; если закопаетесь — не достигнете цели (в реальной жизни это называется «архитектурный паралич» и случается довольно часто);
- кое-какие детали всё-таки надо продумать:
  - как представляются команды и пайпы из команд,
  - как и кем команды создаются,

- как и кем они исполняются,
- как происходит ввод-вывод в пайпе, что с потоком ошибок и кодом возврата,
- кто и как выполняет разбор входной строки,
- кто и как выполняет подстановки (тут особо аккуратно, недостаточно подробное описание стратегии подстановки может дать вам что-то вроде машины Маркова),
- как представляются и кому передаются переменные окружения,
- что с многопоточностью (тут тоже лучше осторожно, потому что потом это реализовывать придётся — однопоточное/однопроцессное решение вполне ок, хоть и отличается от реальных шеллов).