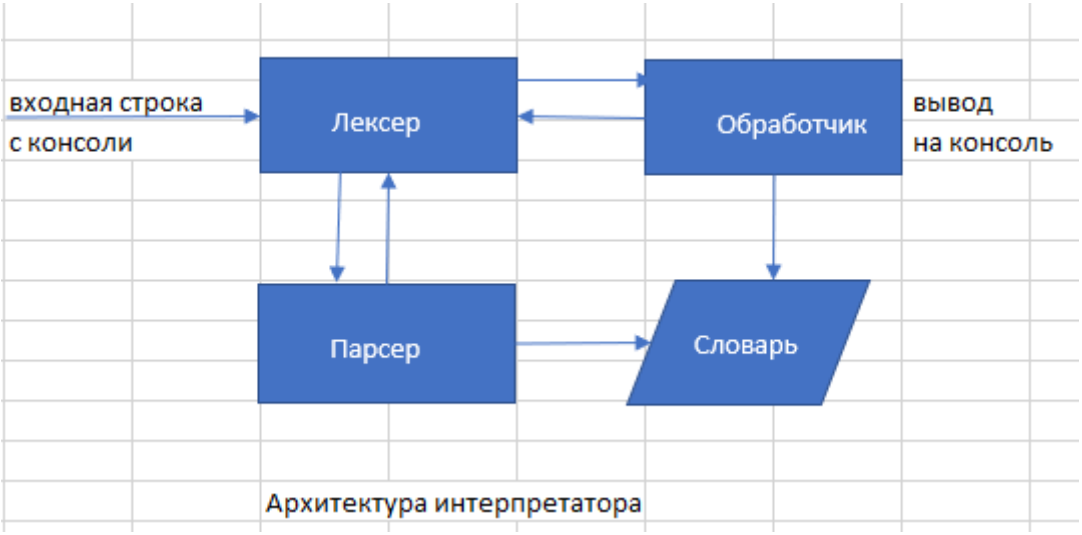
Лексический анализатор (лексер, сканер) - просматривает *литеры* исходной программы слева направо и строит токены. Для краткости будем обозначать его через **Lexer.**

Синтаксический анализатор (парсер) - Задача парсера — провести синтаксический разбор выражения. Работает в паре с **Lexer**.

Обработчик (виртуальная машина) - выполняет полученные команды. Ниже будем использовать обозначение **exec_command**.

Словарь (таблица идентификаторов) – используется для хранения значений переменных (ключ – название переменной, значение – присвоенное ранее значение данной переменной), если на замену вызывается переменная, имя которой отсутствует в словаре, то считаем значение данной переменной пустой строкой.



Архитектура интерпретатора

Далее для краткости будем использовать следующие определения:

Команда – набор токенов (строк), первый элемент набор соответствует непосредственно самой команде, остальные ее аргументы. При реализации команде будет соответствовать вектор (массив) **tokens**.

Буфер – вывод команды при ее исполнении, при реализации данному объекту будет соответствовать переменная **buffer**.

Можно предложить несколько вариантов работы

1) Лексер сначала полностью разбивает строку на команды и передает ее на обработку парсеру, сформированные команды последовательно выполнение обработчик.
2) Как только лексер совместно с парсером сформировали команду, сразу отдает ее на выполнение обработчику.

У каждого варианта есть свои плюсы и минусы. В первом варианте более простая логика работы лексера и парсера по сравнению со вторым вариантом. Но если строка ввода достаточно длинная, содержит много команд, то второй вариант имеет преимущество, он может последовательно по частям обрабатывать входную строку. Сформировал команду и сразу отдал на выполнение. Далее остановимся на реализации второго варианта.

Много команд в одной строке может возникнуть из-за наличия пайплайнов. Можно по-разному обрабатывать пайплайны, мы остановимся на следующем варианте. По условию будем обрабатывать только пайплайны вне кавычек (пайплайн внутри одинаковых кавычек интерпретируется как обычный символ). Первая команда формируется из части входной строки до первого пайплайна, вторая после первого до второго пайплайна (вне кавычек) и т.д.

Команды, получаемы из входной строки выполняем последовательно. Мысленно считаем, что это последовательность строк, введенных на исполнение. В частности, если в первой команде выполнилось присваивание переменной, то это присваивание используется при заменах в следующих командах. Отличие лишь в том, что результат выполнения предыдущей команды (в одной строке) подается на вход следующей (если она предусматривает аргумент).

Условимся, что лишние аргументы игнорируются.

Ниже работа интерпретатора описана более формально.

Описание работы интерпретатора

При вызове интерпретатора в консоли запускается основной цикл, в котором считывается вводимая строка и вызывается **Lexer**.

Lexer

Обработка кавычек

Для обработки кавычек заведем глобальную переменную **open_quote**, которая принимает значения:
0 - нет открытых кавычек;
1 - открыта одиночная кавычка;
2 - открыта двойная кавычка.

При обработки текущего символа "c" входной строки **Lexer** работает по следующим правилам:

если нет открытых кавычек (open_quote == 0), то

если "c" - одиночная кавычка, то **Lexer** устанавливает open_quote = 1,

если "c" - двойная кавычка, то **Lexer** устанавливает open_quote = 2,

если "c" - пайплайн, то **Lexer** вызывает метод парсера **Parser.pipe()**, Команда сформирована, **Lexer** вызывает для ее исполнения обработчик exec_command.

если "c" - символ доллара, то **Lexer** вызывает метод **Parser.replacement**(),

если "c" - символ пробела, то **Lexer** вызывает метод **Parser.space()**,

иначе **Lexer** добавляет данный символ в формируемый токен,

если открыта одиночная кавычка (open_quote == 1), то **Lexer** вызывает метод **Parser.single_quote()**;

если открыта двойная кавычка (open_quote == 2), то **Lexer** вызывает метод **Parser.double_quote()**.

По окончанию обработки строки **Lexer** проверяет флаг open_quote:

если нет открытых кавычек (open_quote == 0), то записывает последний сформированный токен в команду и вызывает обработчик **exec_command** на выполнение сформированной команды.

Далее управление переходит в основной цикл (из которого был вызван **Lexer**).

Описание методов парсера

Parser.replacement() - для обработки замен

метод заводит строковую переменную **var** для имени переменной,

бежит далее по строке и добавляет читаемые символы в **var**, пока не встретит один из символов: пробел, одиночная кавычка, двойная кавычка, знак '=', знак '$' или знак '|'(пайплайна) или пока строка не закончится.

Далее метод ищет в словаре по ключу (равному сформированной переменной **var**) значение (если не находит, то считает его равным "") и добавляет к формируемому токену полученное значение.

Parser.pipe() – обработка пайплайнов

метод добавляет сформированный токен в формируемую команду (массив tokens).

Parser.single_quote() - обработка одиночных кавычек

метод бежит далее по входной строке, пока не встретит одиночную кавычку и добавляет соответствующие символы в формируемый токен. Если встретит одиночную кавычку, то сбрасывает флаг (open_quote = 0).

Parser.double_quote() - обработка двойных кавычек

метод бежит далее по входной строке:

если встречает символ двойная кавычка, то сбрасывает флаг (open_quote = 0) и заканчивает работу,

если встретит символ доллара, то вызывает метод Parser.replacement() и заканчивает работу,

если встретит другой символ, то добавляет его в формируемый токен.

Parser.space() – обработка пробела

если на данный момент сформированный токен не пустой, то метод добавляет его в команду (массив tokens).

Обработчик (exec_command)

Как только команда сформировалась, **Lexer** вызывает **exec_command** для ее выполнения и после исполнения продолжает формировать новые команды.

Если команда предусматривает наличие аргументов, а они отсутствуют, но заполнен **buffer**, то значение из **buffer** рассматривается как аргумент команды.

При выполнении команды exec_command игнорирует наличие лишних аргументов.

Результат выполнения команды записывается в **buffer**. Считаем, что результат всегда есть, но может быть пустым.

После выполнения команды **exec_command** очищает глобальную переменную **Команда** для формирования в нее нового значения.

exit

Работа программы заканчивается, когда на выполнение поступает команда **exit** (не важно с какими аргументами, хотя можно и учесть код ошибки).

Присваивание

Выполнение присваивания осуществляет exec_command. Присваивание обрабатывается только в случае, когда команда содержит знак "=". В этом случае часть до первого вхождения знака "=" считается именем переменной, а то, что после - значением переменной, данная пара записывается в **словарь**.

Если в **словаре** уже в качестве ключа есть такое имя переменной, то ее значение меняется на новое.

Имя переменной должно быть непустое, состоять из букв латинского алфавита, цифр или знака подчеркивания и начинаться не на цифру. Если условие нарушено, то игнорировать выполнение команды и выдавать сообщение "Недопустимое имя переменной".

Все аргументы команды присваивания игнорируются. Результатом команды присваивания является пустая строка.

Для выполнения стандартных команд cat, echo, wc, pwd

Обработчик exec_command вызывает соответствующие функции.

Если команда не опознана (отлична от exit, не содержит знака присваивания и отлична от команд cat, echo, wc, pwd), то она вызывается как внешняя программа, при этом ее вывод направляется во временный файл и потом записывается в buffer.