Федеральное государственное автономное образовательное
учреждение высшего образования
Университет ИТМО

Факультет программной инженерии и компьютерной техники

Отчёт по лабораторной работе №2 по дисциплине «Низкоуровневое программирование»

Вариант: MongoShell

Выполнил:
Деев Роман Александрович

Группа: **P33102**

Преподаватели:
Кореньков Ю. Д.

Санкт-Петербург 2022 г.

Задание:

Задание 2

Использовать средство синтаксического анализа по выбору, реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных. Должна быть обеспечена возможность описания команд создания, выборки, модификации и удаления элементов данных.

Порядок выполнения:

1. Изучить выбранное средство синтаксического анализа
    a. Средство должно поддерживать программный интерфейс совместимый с языком C
    b. Средство должно параметризоваться спецификацией, описывающий синтаксическую структуру разбираемого языка
    c. Средство может функционировать посредством кодогенерации и/или подключения необходимых для его работы дополнительных библиотек
    d. Средство может быть реализовано с нуля, в этом случае оно должно быть основано на обобщённом алгоритме, управляемом спецификацией
2. Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа
    a. При необходимости добавления новых конструкций в язык, добавить нужные синтаксические конструкции в спецификацию (например, сравнения в GraphQL)
    b. Язык запросов должен поддерживать следующие возможности:
        - Условия
            - На равенство и неравенство для чисел, строк и булевских значений
            - На строгие и нестрогие сравнения для чисел
            - Существование подстроки
        - Логическую комбинацию произвольного количества условий и булевских значений
        - В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)
        - Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных
        - Поддержка арифметических операций и конкатенации строк не обязательна
    c. Разрешается разработать свой язык запросов с нуля, в этом случае необходимо показать отличие основных конструкций от остальных вариантов (за исключением типичных выражений типа инфиксных операторов сравнения)
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов
    a. Программный интерфейс модуля должен принимать строку с текстом запроса и возвращать структуру, описывающую дерево разбора запроса или сообщение о синтаксической ошибке
    b. Результат работы модуля должен содержать иерархическое представление условий и других выражений, логически представляющие собой иерархически организованные данные, даже если на уровне средства синтаксического анализа для их разбора было использовано линейное представление
4. Реализовать тестовую программу для демонстрации работоспособности созданного модуля, принимающую на стандартный ввод текст запроса и выводящую на стандартный вывод результирующее дерево разбора или сообщение об ошибке
5. Результаты тестирования представить в виде отчёта, в который включить:
    a. В части 3 привести описание структур данных, представляющих результат разбора запроса
    b. В части 4 описать, какая дополнительная обработка потребовалась для результата разбора, представляемого средством синтаксического анализа, чтобы сформировать результат работы созданного модуля
    c. В части 5 привести примеры запросов для всех возможностей из п.2.b и результирующий вывод тестовой программы, оценить использование разработанным модулем оперативной памяти

Цель

Используя библиотеку для парсинга выражений по придуманной грамматике, создать парсер языка запросов.

Аспекты реализации

Для реализации использован Flex + Bison.

На stdin парсеру подаётся строка запроса. На выходе — структура query, которая описана в файле src/types.h и выводится на stdout.

Типы запросов: insert, find, update, delete. Операторы: $or, $and, $lt, $lte, $gt, $gte, $ne, $regex(для поиска подстроки)

Грамматику языка можно найти в файле src/parse.y. Для построение синтаксического дерева правилах грамматики добавлен код формирующий итоговое дерево.

Результаты

Исходный код: https://github.com/deevroman/low-level-programming-labs

Код реализованного модуля лежит в папке parser.

Кодогенерация и сборка примера с помощью cmake.

Для сборки у себя:

```
git clone https://github.com/deevroman/low-level-programming-labs.git && cd
low-level-programming-labs
# сборка модуля
cmake -B parser/build -DCMAKE_BUILD_TYPE=Release parser
cmake --build parser/build --config Release
# запуск
./parser/build/bin/*/*

При успешной сборке запущенный бинарник будет ожидать ввода.
```

При проблемах обратитесь к файлу `.github/workflows/parser.yml` и к логам GitHub Action, в котором выполняются шаги описанные в файле. Там же можно найти код замера памяти.

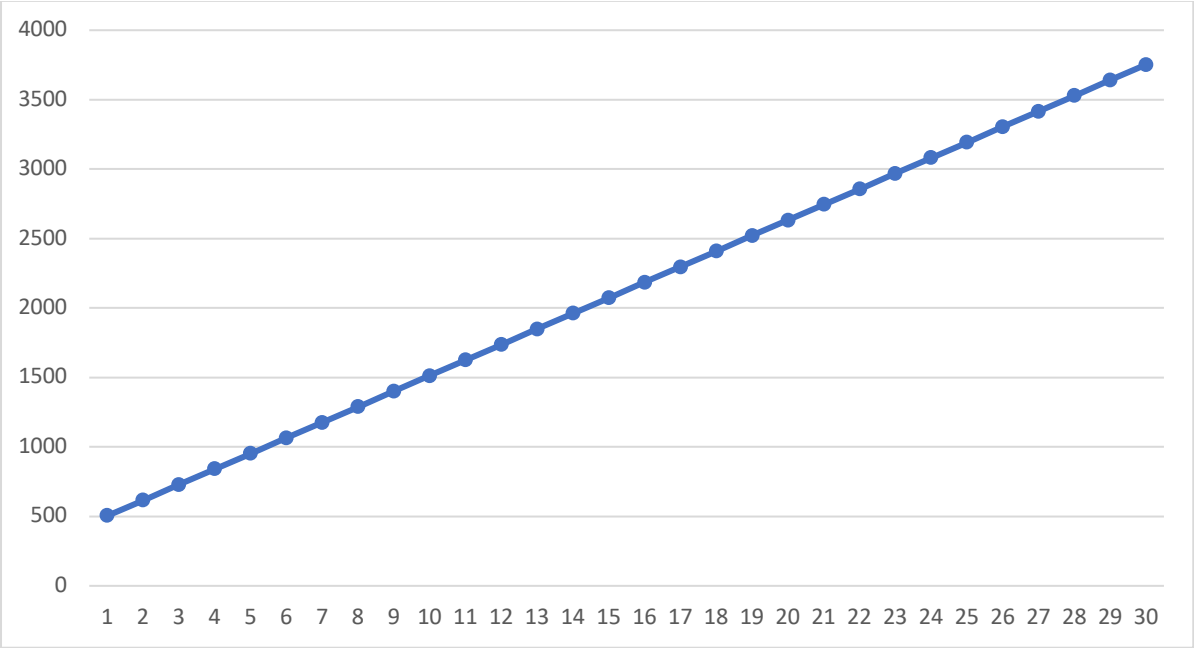
Примеры запросов разбора из файла examples.txt и результат их разбора:

```
db.insert(123, "SCHEMA", {pole:"takoe", vtoroe_pole: True})
Command: Insert
Parent id: 123
Schema: SCHEMA
New fields:
'pole':'takoe'
'vtoroe_pole':'true'
Allocations size: 64 bytes

db.find("SCHEMA", {pole:"takoe", vtoroe_pole: True})
Command: Find
Schema: SCHEMA
Filters:
  &&
  'pole':'takoe'
  'vtoroe_pole':'true'
Allocations size: 168 bytes

db.find("SCHEMA", {pole:"takoe", $or[$and[ipole:"itakoe2", ipole:"itakoe3"],
pole:"takoe3"]})
Command: Find
Schema: SCHEMA
Filters:
  &&
  'pole':'takoe'
    ||
      &&
        'ipole':'itakoe2'
        'ipole':'itakoe3'
      'pole':'takoe3'
Allocations size: 392 bytes

db.update("SCHEMA", {pole:{$ne:"oops"}, vtoroe_pole: True}, {$set: {pole:"oops",
vtoroe_pole: False}})
Command: Update
Schema: SCHEMA
Filters:
  &&
  'pole':'oops'
  'vtoroe_pole':'true'
New fields:
'pole':'oops'
'vtoroe_pole':'false'
Allocations size: 232 bytes

db.update("SCHEMA", {pole:{$ne:"oops"}, vtoroe_pole: True}, {$set: {pole:"oops",
vtoroe_pole: False}})
Command: Update
Schema: SCHEMA
Filters:
  &&
  'pole':'oops'
  'vtoroe_pole':'true'
```

```
New fields:
'pole':'oops'
'vtoroe_pole':'false'
Allocations size: 232 bytes

db.delete("SCHEMA", {pole:"takoe", vtoroe_pole: True})
Command: Delete
Schema: SCHEMA
Filters:
  &&
  'pole':'takoe'
  'vtoroe_pole':'true'
Allocations size: 168 bytes
```

Ресурсоёмкость

Зависимость размера выделенной памяти от количества условий в фильтрах:



Тестировоание происходит на основе запроса из файла `bench.txt` с последовательным добавлением в него новых условий

Выводы:

- Познакомился с Flex и Bison
- Построил грамматику языка запросов
- Реализовал парсер языка запросов.
- ~~Возненавидел macOS за недоделанные консольные утилиты~~