Федеральное государственное автономное образовательное учреждение высшего образования "Национальный Исследовательский Университет ИТМО"
Мегафакультет Компьютерных Технологий и Управления
Факультет Программной Инженерии и Компьютерной Техники

Лабораторная №2
по дисциплине
'Низкоуровневое программирование'

Выполнил Студент группы P33102
Лапин Алексей Александрович
Преподаватель:
Кореньков Юрий Дмитриевич

г. Санкт-Петербург
2023г.

Содержание

1 Цель:

Выданный вариант - 5 (AQL - ArangoDb Query Language)

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.

2 Задачи:

1. Изучить выбранное средство синтаксического анализа
   - Средство должно поддерживать программный интерфейс совместимый с языком C
   - Средство должно параметризоваться спецификацией, описывающий синтаксическую структуру разбираемого языка
   - Средство может функционировать посредством кодогенерации и/или подключения необходимых для его работы дополнительных библиотек
   - 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 возможность описания следующих конструкций: порождение нового элемента данных, выборка, обновление и удаление существующих элементов данных по условию
      - 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

2.1 Выполнение

Для создания лексического анализатора использовался **flex**. Для создания синтаксического анализатора использовался **bison**. Для перехода к новому запросу надо оставить пустую строку. Для завершения программы надо вывести символ EOF (Ctrl+D в терминале).

2.2 Структуры данных

```
  /* Типы вершин AST */
  enum ntype
/* Абстрактное дерево */
struct ast {
    ntype_t nodetype;
    struct ast *l;
    struct ast *r;
};
/* Листья, соответствующих типов данных */
struct nint, struct nfloat, struct nstring

/* Узлы, соответствующие операциям */
/* Операция for */
struct for_ast {
    ntype_t nodetype;
    char* var;
    char* tabname;  // имя таблицы
    struct ast* nonterm_list_head; // список действий
    struct ast* terminal; // завершающее действие
```

```
};

/* Операция filter */
struct filter_ast {
    ntype_t nodetype;
    struct ast* conditions_tree_root; // дерево условий
};

/* Операция insert */
struct insert_ast {
    ntype_t nodetype;
    char* tabname;
    struct ast* list; // список пар, ключ-значение
};

/* Операция update */
struct update_ast {
    ntype_t nodetype;
    char* tabname;
    struct ast* attr; // атрибут
    struct ast* list;
};

/* Операция remove */
struct remove_ast {
    ntype_t nodetype;
    char* tabname;
    struct ast* attr;
};

/* Операция return */
struct return_ast {
    ntype_t nodetype;
    struct ast* value;
};

/* Операция create */
struct create_ast {
    ntype_t nodetype;
    char* name;
    struct ast* difinitions;
};

/* Операция drop */
struct drop_ast {
    ntype_t nodetype;
    char* name;
};

/* Вспомогательные узлы для разных операций */
struct list_ast, struct filter_condition_ast,
struct filter_expr_ast, struct attr_name_ast,
struct merge_ast, struct pair_ast, struct condition_ast,
```

```
73 struct variable_ast, struct create_pair_ast
```

3 Дополнительная обработка для результата разбора

Разобранное выражение на основе правил грамматики преобразуется в вершину AST. Продолжая разбор мы создаем новые вершины дерева, связывая их с уже существующими. После завершения разбора мы получаем дерево, которое можно обойти и вывести в stdout. После создания новой вершины, мы обновляем указатель на корневую вершину, тем самым в любой момент времени мы можем получить корень дерева. Если возникают ошибки, то они попадают в yyerror, который выводит сообщение об ошибке, а также строчку и колонку в stdout. yyerror рекурсивно освобождает память, выделенную под дерево.

3.1 Примеры запросов

3.1.1 Создание таблицы

```
> CREATE users WITH { name: string, lastname: string, student:
   bool, money: int, score: float}
create: {
  tabname: users
  data: [
    definition: {
      name: name
      type: string
    }
    definition: {
      name: lastname
      type: string
    }
    definition: {
      name: student
      type: bool
    }
    definition: {
      name: money
      type: int
    }
    definition: {
      name: score
      type: float
    }
  ]
}
```

3.1.2 Удаление таблицы

```
> DROP users
drop: {
  tabname: users
```

```
}
```

3.1.3 Добавление элемента в таблицу

```
> INSERT { name: "Alex", lastname: "Lapin", student: true, money:
    100, score: 5.0 } INTO users
insert: {
  tabname: users
  data: [
    pair: {
      key: name
      value: {
        string: "Alex"
      }
    }
    pair: {
      key: lastname
      value: {
        string: "Lapin"
      }
    }
    pair: {
      key: student
      value: {
        bool: true
      }
    }
    pair: {
      key: money
      value: {
        int: 100
      }
    }
    pair: {
      key: score
      value: {
        float: 5.0000
      }
    }
  ]
}
```

3.1.4 Удаление элемента из таблицы

```
  > FOR u IN users
  FILTER u.score < 2
  REMOVE u IN users
for: {
  var: u
  tabname: users
  body: [
    filter: {
```

```
        conditions: {
          filter_expr: {
            cmp: <
            attr_name: {
              variable: u
              attrubute: score
            }
            int: 2
          }
        }
      }
    ]
    remove: {
      tabname: users
      attr_name: {
        variable: u
      }
    }
  }
```

3.1.5 Обновление элемента в таблице

```
  > FOR u IN users
  FILTER u.score < 2 AND u.name == "Alex"
  UPDATE u WITH { score: 5 } IN users
for: {
  var: u
  tabname: users
  body: [
    filter: {
      conditions: {
        logic: AND
        filter_expr: {
          cmp: <
          attr_name: {
            variable: u
            attrubute: score
          }
          int: 2
        }
        conditions: {
          filter_expr: {
            cmp: ==
            attr_name: {
              variable: u
              attrubute: name
            }
            string: "Alex"
          }
        }
      }
    }
  ]
```

```
  update: {
    tabname: users
    attr_name: {
      variable: u
    }
    data: [
      pair: {
        key: score
        value: {
          int: 5
        }
      }
    ]
  }
}
```

3.1.6 Выборка элементов из таблицы с условиями

```
  > FOR u IN users
  FILTER u.score > 4 || u.name == "Alex"
 FOR s IN students
  FILTER u.name == s.name && "Alex" IN u.name
  RETURN MERGE(u,s)
for: {
  var: u
  tabname: users
  body: [
    filter: {
      conditions: {
        logic: OR
        filter_expr: {
          cmp: >
          attr_name: {
            variable: u
            attrubute: score
          }
          int: 4
        }
        conditions: {
          filter_expr: {
            cmp: ==
            attr_name: {
              variable: u
              attrubute: name
            }
            string: "Alex"
          }
        }
      }
    }
    for: {
      var: s
      tabname: students
```

```
      body: [
        filter: {
          conditions: {
            logic: AND
            filter_expr: {
              cmp: ==
              attr_name: {
                variable: u
                attrubute: name
              }
              attr_name: {
                variable: s
                attrubute: name
              }
            }
            conditions: {
              filter_expr: {
                cmp: IN
                attr_name: {
                  variable: u
                  attrubute: name
                }
                string: "Alex"
              }
            }
          }
        }
      ]
    }
  ]
  return: {
    merge: {
      variable: u
      variable: s
    }
  }
}
```

4 Запуск

Для запуска программы надо выполнить команду make в папке с проектом. После этого в папке с проектом появится исполняемый файл lab2. Для запуска программы надо выполнить команду ./lab2.

5 Выводы

В ходе выполнения лабораторной работы было изучено средство синтаксического анализа **bison** и **flex**. Был реализован модуль для разбора языка запросов по выбору в соответствии с вариантом формы данных. Была обеспечена возможность описания команд создания, выборки, модификации и удаления элементов данных. Был изучен базовый синтаксис языка AQL.