Курс: **Введение в тензорные компиляторы** Дедлайн: **23 февраля**
Тип: **Задание по теме лекций** Название: **SLR**

Синтаксчиский анализ *shift/reduce* представляет собой разновидность восходящего анализа, в котором для хранения символов грамматики используется стек, а для хранения остающейся непроанализированной части входной строки - входной буффер.

Реализуйте синтаксический анализ простого "арифметического"языка с визуализацией всего процесса, т.е. значений в стеке, входного буффера и совершенного действия (перенос или свертка). На выходе должна быть примерно такая таблица:

STACK	INPUT	ACTION
$	ID1 * ID2$	Shift
$F	* ID2$	Reduce T ->F
$T	* ID2$	Shift
$T*	ID2$	Shift
$T*ID2	$	Reduce F ->ID
$T*F	$	Reduce T ->T*F
$T	$	Reduce E ->T
$E	$	Accept

Таблица 1: Пример вывода программы

- Арифметический язык состоит из сложения, вычитания, умножения, деления и скобок. Операции могут производиться над переменными или же над числами. Имена переменных формируются только из букв латинского алфавита (строчные и заглавные).
- Частью задания является описание грамматики этого языка и любых нетривиальных конструкций, необходимых для парсинга. Таковыми являются, например, детерминированный конечный автомат для LR(0) и таблица парсинга для SLR(1), которые рекомендуется использовать. Опишите это в *README.md* файле.
- Программа должна быть написана на *C* или *C* + +. При этом она должна быть написана хорошо - подумайте, какие иерархия структур/интерфейс функций здесь подойдет. Например (необязательно писать именно так, это лишь пример):

```cpp
// C++ syntax analyzer class example
enum Grammar {
    SLR, // implement only SLR for syntax analyzer
    LALR,
    NONE
};

template <Grammar G>
class SyntaxAnalyzer{};

template<>
class SyntaxAnalyzer<SLR> {
// your implementation
};
```

```c
// C syntax analyzer interface example
typedef enum grammar {
    SLR, // implement only SLR for syntax analyzer
    LALR,
    NONE
} grammar_t;

typedef struct syntax_analyzer {
    // your implementation
} syntax_analyzer_t;

syntax_analyzer_t* syntax_analyzer_init(syntax_analyzer_t* s, grammar_t t) {
    assert(t == SLR && "unsupported grammar type");
    // your implementation
}

//...
```

- Используйте *FLeX* для получения потока токенов. Не стоит придумывать велосипед и парсить токены вручную.
- Необходимо самим релизовать синтаксический анализ именно этого языка, не надо пытаться реализовать общий случай. Так же запрещено использовать *Bison* и аналогичные готовые решения.
- Визуализация вдохновлена драгонбуком (А. Ахо М. Лам Р. Сети Дж. Ульман "Компиляторы: принципы, технологии и инструментарий"). Примерная иллюстрация находится в разделе 4.5.3 Синтаксический анализ "перенос/свертка"
- Уделите внимание деталям - репозиторий должен быть хорошо оформлен. Реализуйте иерархию директорий, например *src*, *include*, *tests*. Используйте систему сборки (*CMake*, *Makefile*, *Bazel*). Напишите инструкцию по сборке и запуску в *README.md*. не забудьте написать тесты на вашу программу.

Для решения задания необходимо сделать репозиторий (*Github*, *Gitlab*, *Gitverse*) и отправить ссылку одному из преподавателей в Телеграме (@synthmoza @vloznovenko)