

Формальные языки

Проект для Находько Артёма

1. Реализовать синтаксический анализатор для языка арифметических выражений (подмножество базового синтаксиса, описанного ниже).

- В вашем случае корректный вход состоит из одного арифметического выражения.
- Можно использовать любой способ писать парсер.
- Для языка предложить свой конкретный синтаксис. Описание конкретного синтаксиса прислать не позднее 20 декабря.
- Реализацию парсера прислать не позднее 23:59 9 января.
- Не позднее дня перед экзаменом надо будет созвониться, чтобы вы мне рассказали, что как работает. Созваниваться будем после того, как будет зачтен код.
- Скорее всего потребуется несколько итераций проверки, не затягивайте.
- В конце файла требования к оформлению, их надо соблюдать.

Описание базового синтаксиса

- Лексический синтаксис

- Идентификатор — непустая последовательность букв латинского алфавита в любом регистре, цифр и символа нижнего подчеркивания (`_`), начинающаяся на букву латинского алфавита в нижнем регистре, не являющаяся ключевым словом.
 - * Корректные идентификаторы: `x`, `list`, `listNat_123`.
 - * Некорректные идентификаторы: `Abc`, `123`, `_List`.
- Число: натуральное или ноль в десятичной системе счисления, не может содержать лидирующие нули.
 - * Корректные числа: `123`, `0`.
 - * Некорректные числа: `-1`, `007`, `89A`.
- Ключевые слова не могут быть идентификаторами. Конкретные ключевые слова вы выбираете сами.
- Операторы языка:
 - * сложение `+`,
 - * умножение `*`,
 - * деление `/`,
 - * вычитание `-`,
 - * возведение в степень `**`,
 - * конъюнкция `&&`,
 - * дизъюнкция `||`,
 - * логическое отрицание `--`,
 - * операторы сравнения: `<`, `<=`, `==`, `<>`, `>`, `>=`,
- Пробелы не являются значимыми, но не могут встречаться внутри одной лексемы.

- Базовый абстрактный синтаксис

- Программа — непустая последовательность определений функций.
- Определение функции содержит ее сигнатуру и тело. Сигнатура функции содержит ее название (идентификатор) и список аргументов (может быть пустым). Тело — последовательность инструкций (может быть пустой).
- Инструкции:
 - * Присвоение значения арифметического выражения переменной. Переменная может быть произвольным идентификатором.
 - * Возвращение значения из функции.
 - * Условное выражение с обязательной веткой **else**. Условием является арифметическое выражение. В ветках — произвольные последовательности инструкций (могут быть пустыми).
 - * Цикл с предусловием. Условием является арифметическое выражение. Тело цикла — произвольная последовательность инструкций (может быть пустой).
- Арифметические выражения заданы над числами и идентификаторами, операторы перечислены в таблице ниже с указанием их приоритетов, аргументности и ассоциативности.

Наибольший приоритет	Аргументность	Ассоциативность
-	Унарная	
**	Бинарная	Правоассоциативная
*,/	Бинарная	Левассоциативная
+, -	Бинарная	Левассоциативная
==, <>, <, <=, >, >=	Бинарная	Неассоциативная
--	Унарная	
&&	Бинарная	Правоассоциативная
	Бинарная	Правоассоциативная
Наименьший приоритет	Аргументность	Ассоциативность

Требования к оформлению

- Результатом должно быть консольное приложение, которое принимает на вход программу и печатает результат синтаксического анализа в файл с таким же названием и дополнительным расширением `.out`. (`input.txt` → `input.txt.out`)
- Результатом синтаксического анализа является абстрактное синтаксическое дерево в случае успешного разбора и сообщение об ошибке иначе.
 - Если произошла лексическая ошибка, то сообщить о ней и завершиться, не пытаясь парсить дальше.
 - Если произошла любая ошибка — сообщить о ней и завершиться, не пытаясь восстанавливаться после ошибки.
- Код может быть написан на любом языке программирования с использованием любых инструментов, но должен быть сопровожден инструкцией по сборке и запуску. Желательно выложить его на гитхаб и сопроводить тестами.
- Ссылку на репозиторий присылать в письме с темой `[FL_ElTech] project`.