# ФПМИ МФТИ.

## Теория формальных языков.
## Практикум. Группы 022, 023

*Дедлайн: 12 декабря 20:59*

*Вы решаете один из двух вариантов практикума, который влияет на дальнейшее продвижение по курсу:*

- Практический вариант – дает баллы за практику (от 0 до 7).
- Интерпретатор – дает баллы за практикум + **вступительное задание на курс по компиляторам весной**. Сдаётся лектору или семинаристу, к которому он вас отправит. Если вы выбрали этот вариант, обязательно свяжитесь с лектором, узнайте у него, кому вам сдавать задание и задавать вопросы. Также сообщите мне, что вы делаете этот вариант практикума.

*Если вы выбираете первый вариант, надо добавить в таблицу ссылку на репозиторий, вопросы можно задавать в чате в Telegram. Если вы выбираете второй вариант, надо написать на почту formal.2021.hw.akhtyamov.practice@akht.pl, сообщить об этом и узнать, как и кому вы сдаёте задание и куда можно задавать вопросы. Важно! В теме письма указывайте* **[Interpreter] FirstName LastName, GroupNumber**.

*Практическое задание*

Необходимо реализовать алгоритм в виде класса, в котором есть следующие методы:

- `fit(G: Grammar)` → `Algo` - препроцессинг (в алгоритме Эрли можно просто запоминать грамматику). В случае $LR(1)$ если грамматика не является $LR(1)$ грамматикой, должно бросаться исключение.
- `predict(word: String)` → `Boolean` - проверка принадлежности слова языку.

Алгоритмы для реализации

1. Алгоритм Эрли
2. LR(1)-алгоритм

Если вы реализуете только алгоритм Эрли, вы получите за практикум не более 2 баллов из 7. Если вы реализуете только алгоритм LR(0), вы получите за практикум не более 4 баллов из 7. В алгоритме Эрли необходимо реализовать функции:

- `Scan(conf: Configuration, letter: char)` → `Set<Configuration>`
- `Predict(...)`
- `Complete(...)` - продумайте интерфейс таким образом, чтобы результат можно было протестировать.

Нужно реализовать алгоритмы в виде классов и написать программу, которая считывает грамматику, строит по ней анализатор, считывает слова, проверяет, лежат ли они в языке, и выводит для каждого слова, лежит ли оно в языке. Если входные данные не соответствуют формату входных данных или не задают КС грамматику, программа должна бросать исключение.

Очень рекомендуется реализовать тестирование алгоритма. Если вы пишите тесты, добавляйте их в репозиторий. Тестирование не будет проверяться, но будет проверяться, что алгоритм работает корректно. Если я смогу придумать корректные входные данные, на которых ваша программа выдаёт неправильный ответ, то вы получите за задание мало баллов (насколько мало зависит от того, насколько существенной мне покажется ошибка, но точно не больше половины, возможно за исключением ситуации, когда ошибка была в функциях для ввода и вывода, а не в классе с алгоритмом).

Дополнительно для ускорения проверки нужно добавить в корневую папку репозитория файлы lr_in_one_file и erly_in_one_file, содержащие программу в одном файле (обычно чтобы получить такую программу надо заменить в файле с main import/include на копипаст кода из импортируемых файлов). Это поможет мне быстрее запустить ваш код и проверить, что он работает, так как скопировать код из одного файла быстрее, чем скачивать репозиторий и собирать код в нескольких файлах. **Это не отменяет того, что надо написать и залить в репозиторий нормально структурированный код, эти файлы нужно добавить дополнительно к нормальному коду, а не вместо него.**

**Формат входных данных**

В первой строке содержатся 3 числа $|N|, |\Sigma|$ и $|P|$ — количество нетерминальных символов, терминальных символов и правил в порождающей грамматике. Все числа неотрицательные и не превосходят 1000.

Во второй строке содержатся $|N|$ нетерминальных символов. Нетерминальные символы являются заглавными латинскими буквами.

В третьей строке содержатся $|\Sigma|$ символов алфавита. Символы являются строчными латинскими буквами, цифрами, скобками или знаками арифметических операций.
В каждой из следующих $P$ строк записано одно правило грамматики в формате левая часть правила -> правая части правила. $\varepsilon$ в правой части правила обозначается отсутствоем правой части (концом строки после ->).
Следующая строка состоит из одного нетерминального символа — стартового символа грамматики.
Следующая строка состоит из одного числа $m$ — количества слов, принадлежности которых языку надо проверить.
В каждой из следующих $m$ строк содержится одно слово, состоящее из символов алфавита грамматики, принадлежность которого языку надо проверить.

**Формат входных данных**
Выведите $m$ строк. $i$-ая строка должна содержать слово "Yes"(без кавычек), если $i$-ое слово из входных данных лежит в языке, пораждаемом грамматикой, и "No"(без кавычек) иначе.

Пример входных и выходных данных

| Вход | Выход |
|---|---|
| 1 2 2<br>S<br>ab<br>S->aSbS<br>S-><br>S<br>2<br>aababb<br>aabbba | Yes<br>No |

*Практическое задание: интерпретатор*

*Описание этого задание полностью скопировано из домашнего задания групп 022 и 023.*
Реализуйте интерпретатор языка программирования на выбор:

- Fortran (можно использовать диалект 77-го, 90-го или 95-го года)
- LOLCODE
- Pascal
- Свой на выбор

В качестве реализации необходимо предоставить запуск программы в виде опций:

`./executable <path/to/file>`

Необходимые части в реализации:

- Арифметика
- Логический функционал (and, or, not)
- Вывод на экран
- if-then-else
- Циклы (for, while)
- Строки

Необходимый старт для проекта можно найти по ссылке: https://github.com/akhtyamovpavel/CompilersC
parsers
Вашей целью будет написание:

- Написание файла `scanner.l`, в котором происходит разделение символов на слова (токены). Токены задаются в виде регулярного выражения (строки 25-27) или набора строк/идентифика (строки 41-60), при этом при помощи `make_TOKEN` задается `TOKEN`, который будет необходим для дальнейшего построения грамматики
- Написание файла `parser.y`, в котором происходит описание грамматики при помощи токенов:
  - Сначала идет forward-declaration необходимых классов
  - Затем идут необходимые include-ы из других файлов
  - `%token` задает отображение из построенных имен токенов в строки, используемые для построения правил грамматики
  - В строке 55 начинается построение грамматики
  - В строке 56 задается стартовый символ
  - Далее идут правила: нетерминал: правила вывода (в фигурных скобках код, который задают обработчики правила). Важно обратить внимание, что используемый алгоритм – LR, поэтому обработка правил идет с нижней части дерева разбора.
- Написание драйвера `Driver.cpp`, в котором описывается логика интерпретации.
- Написание вспомогательных классов, используемых для интерпретации (подумайте, почему наивно все делать не получится)

Важно подчеркнуть, что нет лишних уровней вложения, весь код работает только в одном уровне вложенности (переменные объявляются только на самом верхнем уровне)
Дополнительные возможности построения грамматик вы можете посмотреть в других папках репозитория.
Полезные ссылки:

- https://github.com/justinmeza/lolcode-spec/blob/master/v1.2/lolcode-spec-v1.2.md – спецификация LOLCODE.
- http://courses.washington.edu/css448/zander/Project/grammar.pdf – грамматика для Паскаля
- https://pages.mtu.edu/ shene/COURSES/cs201/NOTES/F90-Basics.pdf – презентация для Фортрана.
- https://learnxinyminutes.com/docs/fortran95/ – learn Fortran.
- https://learnxinyminutes.com/docs/pascal/ – learn Pascal.
- https://learnxinyminutes.com/docs/LOLCODE/ – learn LOLCODE.