1

Курс простой. По нему зачёт.
Нужно решить 4 лабораторные работы:

- Регулярные выражения в языке **Perl**. Самостоятельное изучение материала и сдача в PCMS. Если сдавать в 2022, то можно решить N-3 задач. До конца сессии -2. После -1 (все, кроме одной).
- До 22:00 в пн нужно будет записываться на сдачу лабораторных 2,3,4. Показываешь, что работает. Получаешь модификацию (их много рарных на одно и то же задание). Потом показываешь уже модификацию.
- ЛР2 — ручное построение нисходящих парсеров
- ЛР3 — использование автоматических генераторов парсеров. ANTLR, Happy, Bison (YACC)
- ЛР4 — написать автоматический генератор парсеров

Компиляция: программа → парсинг → генерация исполнимого кода.
Здесь пройдём первый основы парсинга (довольно глубоко), а генерацию кода только базовую основу.
Если вы пойдёте в магистратуру, там будет курс компиляторов. Там фокус наоборот на генерации.

В курсе нет дедлайнов. НО удачи вам с тайм-менеджментом без них :v) Курс по софт-скиллам, you're welcom

Что такое пасрер?

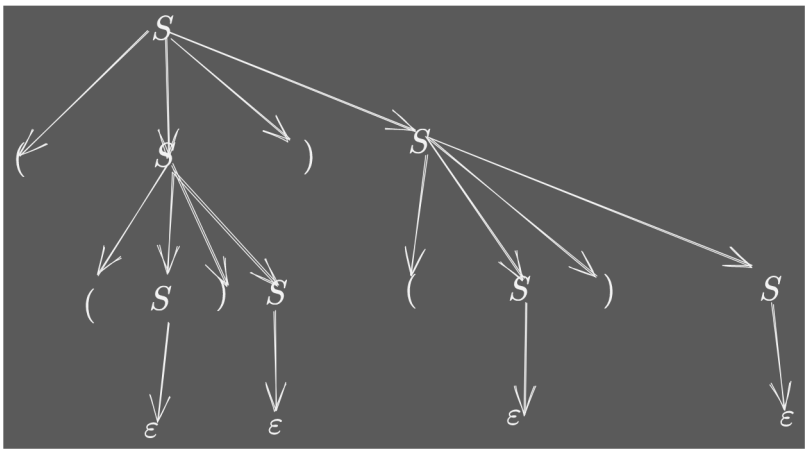
текстовое представление информации → внтреннее представление компьютера. Чаще всего дерево разбора в некоторой контекстно-свободной грамматике.

Регулярное выражение — способ записать регулярные языки.

Контекстно свободная грамматика — алфавит, множество нетерминалов, стартовый нетерминал и правила.

```
неоднозначная:
S → (S)
S → SS
S → e

однозначная:
S → (S)S
S → e
```



Токенизация: инпут в *любом* формате → $c_1, c_2, \ldots, c_n$, где c_i это токены.

Синтаксический/Лексический анализ (парсинг)

Лексический анализатор:
Есть входной алфавит парсера. Есть совсем входной алфавит A (ASCII условно).
Пример: арифметические выражения $\Sigma = \{+, -, *, (,), number\}$.
Числа здесь не нужны. 2+3, 5+5, 17+231. Всё это считается одним выражением. Поэтому есть один токен для числа.

```
+ +
- -
* *
) )
( (
n (1|2|3|4|5|6|7|8|9)(0|1|2| ... |9)*|0
```

```
17+231
n
 n
 n+
 n+n
 n+ n
 n+  n
```

Задача: у вас есть ЯП довольно мощный. Вы прочитали токен. Вам нужно выдать номер токена. Чтобы делать это быстро нужно Perfect < ... > Mapping

---

Парсеры

Алгоритм Кока-Янгера-Касами
Работает только для грамматик в нормальной форме Хомского (A → BC). Можно снять ограничение, но это неприятно. Работает за куб от длины слова. Если бы мы парсили за куб ... было бы очень грустно

```
dp_A[l][r] – можно ли получить s[l..r] из A
```

Парсить хочется быстро.

Снизу вверх vs Сверху вниз.
Направление построения дерева. Его можно строить от корня или от листьев (спрашивая какой у них может быть родитель)

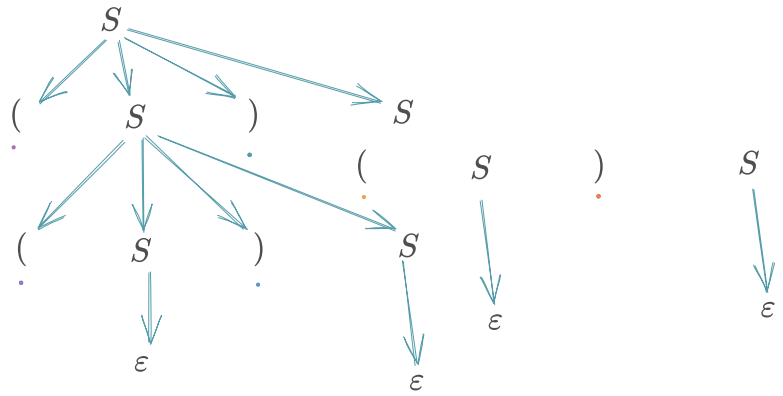
Рекурсивный парсер сверху вниз = рекурсивный спуск.
Есть языки, которые так не распарсить.

^ это было введение

Нисходящие методы разбора

```
S → (S)S
S → e

(())()
```



LL(1) грамматика – знание первого нераскрытого терминала и первого терминала даёт понять следующий шаг разбора.
$S \Longrightarrow {}^*xA\alpha \Longrightarrow x\xi\alpha \Longrightarrow {}^*xc\alpha' \Longrightarrow {}^*xcy*$
$S \Longrightarrow {}^*xA\beta \Longrightarrow x\eta\beta \Longrightarrow {}^*xc\beta' \Longrightarrow {}^*xcz*$

LL(1) грамматика – Если есть два таких вывода, то $\xi = \eta$

Патч: все символы уже подвесили, а первый нераскрытый терминал всё ещё есть. Нужно раскрывать в ε..

Добавим символ конец ввода $

Сейчас определение выше рассматривает бесконечное число выражений (импликаций). Можно сделать, чтобы конечное, об этом позже.

1

> **Методы Трансляции с 2022-09-06**