# Лексический анализ

## Задание для: Поляков Игорь Андреевич

**|** Версия: 2

## Требования, начиная с этой лабы и далее

1. На C# пишем в Microsoft Visual Studio версия минимум 2017 (она существует только под Windows, следовательно должна быть винда; Visual Studio Code и Visual Studio for Mac это совершенно другие продукты).
2. Комбинации клавиш в студии должны быть настроены по умолчанию.
3. Кнопки F5, F10, F11, F12 и т.п. должны физическими кнопками и работать без Fn (настраивается в биосе, если не получится найти, на паре могу помочь).
4. Используем .NET Framework версии минимум 4.5, никакого .NET Core.
5. Стиль кода:
   1. У if, while, foreach и т.п. должны быть фигурные скобки; писать всё в одну строчку нельзя.
      1. Исключение: else можно писать без фигурных скобок, если сразу за ним идет if (речь про: if (a) { ... } else if (b) { ... } else { ... }).
   2. else, catch, finally и т.п. с новой строки.
   3. Цикл do { ... } while не используем.
   4. Никаких пустых строк внутри методов. Если хотите сгруппировать код, можно использовать регионы (#region) или область видимости (block statement, т.е. код внутри { }).
   5. Никаких комментариев, особенно комментариев с описанием того, что делает программа (должны помнить, а лучше на ходу уметь объяснять).
      1. Комментарии можно использовать только чтобы временно закомментировать строку кода в процессе разработки или если они требуются в задании.
      2. Чтобы "выключить" кусок кода используйте другие методы, не ломающие встроенные в студию рефакторинги, например переименование Main или if (false) { ... }.
6. Исходный код должен быть отформатирован с помощью Ctrl-K-D.
7. Вы должны уметь объяснить, что делает любая строчка в вашей программе.

## Часть 1. Лексер

Нужно написать программу, которая делает лексический анализ своего кода. В проекте должен быть только один файл с C# кодом.

* Копируете код и текстовые файлы из первой лабы. Делаете так, чтобы код из первой лабы не вызывался (но можно было быстро включить его и запустить).
* Далее программа:
  + читает свой собственный код;
  + разбивает его на последовательность токенов методом static IEnumerable<Token> GetTokens(string sourceText);
  + выводит в консоль лексемы вместе с типом циклом foreach (это вам для отладки);
  + выводит в выходной файл (сделайте так, чтобы можно было быстро переключаться, меняя код):
    - рядом с каждой лексемой в комментарии прописать тип токена (например: /\*Identifier\*/using/\*Whitespaces\*/ /\*Identifier\*/System/\*Punc\*/./\*Identifier\*/Text и т.д.);
    - лексемы, разделенные пробелом;
    - конкатенацию лексем (должен получиться в точности входной файл);
* Выходной файл должен быть валидной программой в отдельном проекте; если этот проект запустить, он должен перезаписать свой исходный код.
* Так как пишем лексер для подмножества C#, старайтесь использовать фичи языка по минимуму:
  + используйте только английские буквы **для идентификаторов, в строках пишем так, как требует задача**;
  + не используйте типы токенов, без которых можно обойтись (например 'a');
  + все строки должны быть с собачкой (строки без собачки сложно правильно разобрать, а строки с долларом вообще невозможно разобрать регулярками).
* В регулярке явно описываем, какие символы нам нужны (никаких \w, . и т.п.)
* Регулярку старайтесь писать как можно проще, но так как она все равно получится большая, сделайте её читабельной (RegexOptions.IgnorePatternWhitespace или соберите регулярку из нескольких строк).
* **Лексер не должен уметь распознавать лексемы, которых нет в C#. В некоторых случаях буду просить оставить в лексере только используемые во входных данных токены или поменять программу так, чтобы она не использовала какие-то токены.**
* При кривых входных данных должно явно кидаться исключение, т.е. пропускать нераспознанные куски нельзя. **Сообщения в исключениях пишите по-русски, если не можете свободно писать по-английски, но язык должен быть одинаковым во всей программе.**
* Максимально (но без фанатизма) используйте Enumerable.Select, string.Concat, string.Join и т.п.
* **Код из первой лабы должен работать так же, как и в первой лабе.**

Рекомендуемая структура решения (в нем 2 проекта, можно переключаться с помощью "назначить запускаемым проектом по умолчанию" в контекстном меню проекта или слева от кнопки запуска с зеленым треугольником):

* Lab2\
  + Lab2.sln
  + Lab2\
    - Lab2.csproj
    - Program.cs - входной файл с программой
    - файлы из первой лабы
  + Lab2Out\ - создаете этот проект после создания первого
    - Lab2Out.csproj
    - Program.cs - выходной файл
    - файлы из первой лабы

Чтение и запись будут выглядеть как-то так:

var input = File.ReadAllText(@"../../Program.cs");  
...  
File.WriteAllText(@"../../../Lab2Out/Program.cs", output);