

Обработка текста конечными автоматами

Лекция №2

Закон дырявых абстракций

Любая нетривиальная абстракция содержит дыры, сквозь которые видны нижележащие абстракции.

Другими словами: рано или поздно потребуется узнать, какая реализация скрывается за абстракцией.

Постановка задачи

Постановка задачи

Пишем два примера на C#:

1. Отсортировать строки в файле
2. Извлечь список слов из текста

Реализация тут: [Шаблон проекта на C# / .NET](#)

Постановка задачи

Пишем два примера на C#:

1. **Отсортировать строки в файле**
2. Извлечь список слов из текста

```
public static void SortFileLines(string path)
```

Реализация тут: [Шаблон проекта на C# / .NET](#)

Файлы

Что такое файл?

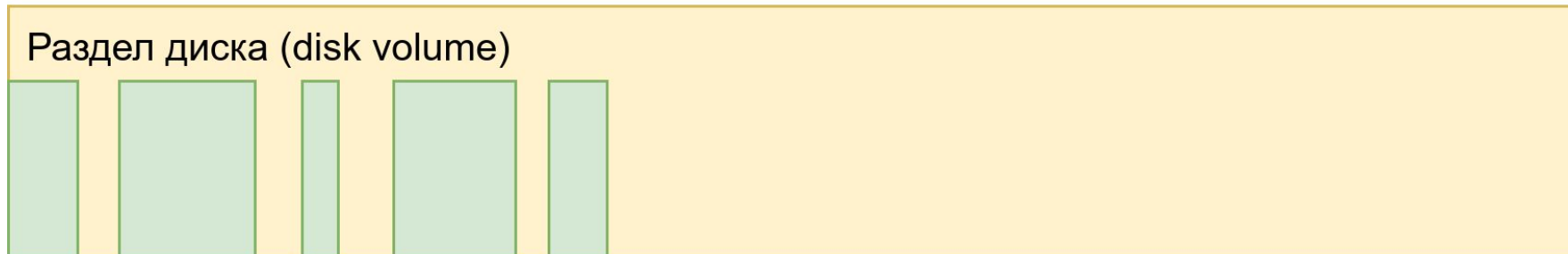
Файл — последовательность байтов с уникальным идентификатором

Что такое файл?

Файл — последовательность байтов с уникальным идентификатором

Файловая система (ФС) — B+ дерево файлов, построенное на разделе диска

 Область, используемая ФС



Файловая система в Linux

Данные файла:

- *логически* — непрерывная последовательность байт
- *физически* — последовательность страниц по 4096 байт

Файловая система в Linux

Данные файла:

- *логически* — непрерывная последовательность байт
- *физически* — последовательность страниц по 4096 байт

Файлы и каталоги — узлы дерева (inode).

Атрибуты файлов и каталогов:

- Имя
- Родитель (parent inode)
- Даты создания / изменения / доступа
- Unix permissions

Работа с файлами в Linux и C#

Режимы открытия файла в Linux

- **O_CREAT** — если файл не существует он будет создан

См. [man open \(2\)](#)

Режимы открытия файла в Linux

- **O_CREAT** — если файл не существует он будет создан
- **O_EXCL | O_CREAT** — если файл существует, вернуть ошибку

См. [man open \(2\)](#)

Режимы открытия файла в Linux

- **O_CREAT** — если файл не существует он будет создан
- **O_EXCL | O_CREAT** — если файл существует, вернуть ошибку
- **O_TRUNC** — сбросить данные файла

См. [man open \(2\)](#)

Режимы открытия файла в Linux

- **O_CREAT** — если файл не существует он будет создан
- **O_EXCL | O_CREAT** — если файл существует, вернуть ошибку
- **O_TRUNC** — сбросить данные файла
- **O_APPEND** — добавлять данные в файл

См. [man open \(2\)](#)

Режимы открытия файла в Linux

- **O_CREAT** — если файл не существует он будет создан
- **O_EXCL | O_CREAT** — если файл существует, вернуть ошибку
- **O_TRUNC** — сбросить данные файла
- **O_APPEND** — добавлять данные в файл
- **O_RDONLY, O_WRONLY, O_RDWR** — режимы для чтения, для записи и для чтения/записи

См. [man open \(2\)](#)

Чтение файла (C++, Linux)

```
int fd = open("input.txt", O_RDONLY);  
if (fd == -1) throw [...];  
  
ssize_t bytes_read = read(fd, buf, sizeof(buf)-1);  
if (bytes_read > 0) {  
    buf[bytes_read] = '\0';  
    // обработка данных  
}  
close(fd);
```

Чтение файла (C#)

```
using FileStream fs = new(  
    "input.txt", FileMode.Open, FileAccess.Read  
);  
  
byte[] buffer = new byte[1024];  
  
int bytesRead = fs.Read(buffer, 0, buffer.Length);  
  
string content = Encoding.UTF8.GetString(  
    buffer, 0, bytesRead  
);
```

Запись строки в файл (C++, Linux)

```
int fd = open(
    "output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644
);
if (fd == -1) throw [...];

const char* msg = "Hello, world!";
write(fd, msg, strlen(msg));
close(fd);
```

Запись строки в файл (C#)

```
const string message = "Hello, world!";  
  
using FileStream fs = new(  
    "output.txt", FileMode.Create, FileAccess.Write  
);  
  
byte[] data = Encoding.UTF8.GetBytes(message);  
fs.Write(data, 0, data.Length);
```

Создание временного файла (C++, Linux)

```
char name[] = GetRandomFileName(  
    "/tmp/tempfile.XXXXXX"  
);  
int fd = open(name, O_RDWR | O_CREAT | O_EXCL, 0644);  
if (fd == -1) throw [...];  
  
write(fd, data, strlen(data));  
close(fd);
```

Создание временного файла (C#)

```
string tempFileName = Path.Combine(  
    Path.GetTempPath(), Path.GetRandomFileName()  
);  
  
using FileStream fs = new(  
    tempFileName, FileMode.CreateNew, FileAccess.ReadWrite  
);
```

Высокоуровневые методы в C#

// 1. Чтение строки из input.txt

```
string content = File.ReadAllText("input.txt");
```

// 2. Запись строки в output.txt

```
File.WriteAllText("output.txt", "Hello, world!");
```

// 3. Создание временного файла

```
string tempFilePath = Path.GetTempFileName();
```

```
File.WriteAllText(tempFilePath, "Temporary content");
```

Сортировка строк файла

Чтение и сортировка строк

```
public static void SortFileLines(string path) {  
    // Целиком читаем файл в память.  
    List<string> lines = File.ReadLines(  
        path, Encoding.UTF8  
    ).ToList();  
    lines.Sort();  
    // [...]
```

Перезапись файла

```
using FileStream file = File.Open(  
    path, FileMode.Truncate, FileAccess.Write  
);  
for (int i = 0, iMax = lines.Count; i < iMax; ++i) {  
    file.Write(Encoding.UTF8.GetBytes(lines[i]));  
    if (i != iMax - 1) {  
        file.Write("\n"u8);  
    }  
}
```

Пишем модульный тест

Модульный тест

[Fact]

```
public void CanSortTextFile() {  
    // [...]  
    using TempFile file = TempFile.Create(unsorted);  
    FileUtil.SortFileLines(file.Path);  
    string actual = File.ReadAllText(file.Path);  
    Assert.Equal(sorted.Replace("\r\n", "\n"), actual);  
}
```

Временные файлы для тестов

```
public sealed class TempFile : IDisposable {  
    private TempFile(string path) {  
        Path = path;  
    }  
  
    public string Path { get; }  
  
    // [...]
```

Временные файлы для тестов

```
// [...]
```

```
public static TempFile Create(string contents) {  
    string path = System.IO.Path.GetTempFileName();  
    File.WriteAllText(path, contents, Encoding.UTF8);  
    return new TempFile(path);  
}
```

```
// [...]
```

Временные файлы для тестов

```
// [...]
```

```
public void Dispose() {
```

```
    File.Delete(Path);
```

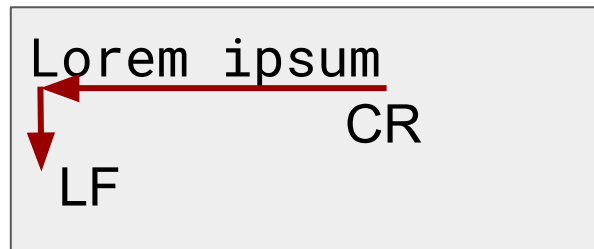
```
}
```

Текстовый файл

Концы строк

Два символа:

1. LF (line feed, ' `\n` ') — перевод на следующую строку
2. CR (carriage return, ' `\r` ') — возврат в начало строки



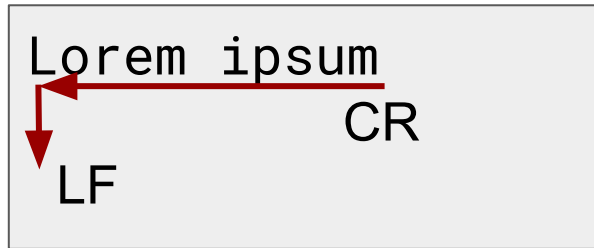
Концы строк

Два символа:

1. LF (line feed, ' `\n` ') — перевод на следующую строку
2. CR (carriage return, ' `\r` ') — возврат в начало строки

Три типа концов строки в файлах:

- ' `\n` ' (LF) — стиль Unix и Linux
- ' `\r\n` ' (CRLF) — стиль MS DOS и Windows
- ' `\r` ' (CR) — стиль старой Mac OS



Стандарт Unicode

- UTF-8, UTF-16, UTF-32 — кодировки стандарта Unicode

Стандарт Unicode

- UTF-8, UTF-16, UTF-32 — кодировки стандарта Unicode
- Символ Unicode — ровно 21 бит, от 0x000000 до 0x10FFFF

Стандарт Unicode

- UTF-8, UTF-16, UTF-32 — кодировки стандарта Unicode
- Символ Unicode — ровно 21 бит, от 0x000000 до 0x10FFFF
 - 17 плоскостей по 65536 символов = 1 114 112 символов
 - Плоскость BMP — символы с номерами от 0 до 0x00FFFF
 - Символы с 0x000000 по 0x00007F совпадают с ASCII

Кодировки Unicode

Кодировка	Размер codepoint	Размер символа
UTF-8	1 байт	1, 2, 3 или 4 байта
UTF-16	2 байта	2 или 4 байта
UTF-32	4 байта	4 байта

Кодировки Unicode

Кодировка	Размер codepoint	Размер символа
UTF-8	1 байт	1, 2, 3 или 4 байта
UTF-16	2 байта	2 или 4 байта
UTF-32	4 байта	4 байта

UTF-16 и UTF-32 имеют версии BE / LE (меняется порядок байт)

Число	Байты в Big Endian	Байты в Little Endian
0x0410	0x04 0x10	0x10 0x04

Кодировка UTF-32

Один символ — 4 байта (32 бита)

- номер символа помещается в 21 бит
- остальные биты заполняются нулями

Кодировка UTF-8

Номер символа	Байт №1	Байт №2	Байт №3	Байт №4
0x0000...0x007F	0xxxxxxx	—	—	—
0x0080...0x07FF	110xxxxx	10xxxxxx	—	—
0x0800...0xFFFF	1110xxxx	10xxxxxx	10xxxxxx	—
0x10000...0x10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8 совместим с ASCII

Кодировка UTF-16

Номер символа	Байт №1	Байт №2	Байт №3	Байт №4
U+0000...FFFF	xxxxxxxx	xxxxxxxx	—	—
U+FFFF...10FFFF	110110xx	xxxxxxxx	110111xx	xxxxxxxx

codepoint (2 байта)

codepoint (2 байта)

Кодировка UTF-16

Номер символа	Байт №1	Байт №2	Байт №3	Байт №4
U+0000...FFFF	xxxxxxxx	xxxxxxxx	—	—
U+FFFF...10FFFF	110110xx	xxxxxxxx	110111xx	xxxxxxxx

codepoint (2 байта)

codepoint (2 байта)

В Unicode символы 0xD800...0xDFFF зарезервированы для UTF-16

BOM (byte order mark)

Символ 0xFEFF — позволяет отличить порядок байт

UTF-8	EF BB BF
UTF-16 (BE)	FE FF
UTF-16 (LE)	FF FE
UTF-32 (BE)	00 00 FE FF
UTF-32 (LE)	FF FE 00 00

Что такое текстовый файл?

1. Не каждый набор байт является текстом в Unicode
 - Текстовый файл — последовательность байт, которая является текстом
 - Бинарный файл — последовательность байт, которая может и не быть текстом
2. В начале файла может быть BOM (byte order mark)
3. Концы строк могут быть разными — '\n', '\r\n', '\r'

Обработка текста

Постановка задачи

Пишем два примера на C#:

1. Отсортировать строки в файле
2. **Извлечь список слов из текста**

```
static List<string> ExtractWords(string text)
```

Реализация тут: [Шаблон проекта на C# / .NET](#)

Unicode в C#

Unicode в C#

Строки в C# — в кодировке UTF-16

- Тип `char` в C# — это не символ, а `codepoint`
- Тип `Rune` в C# — символ Unicode

Unicode в C#

Строки в C# — в кодировке UTF-16

- Тип `char` в C# — это не символ, а `codepoint`
- Тип `Rune` в C# — символ Unicode

Советы:

1. Поддерживайте UTF-8, совместимый с ASCII
2. Принимайте любые концы строк
3. При записи используйте `'\n'` либо `Environment.NewLine`

Итерация по символам Unicode

```
const string text = "Привет, мир!";

foreach (Rune ch in text.EnumerateRunes()) {

    // Rune – символ Unicode (4 байта)

}

foreach (char ch in text) {

    // char – это codepoint в UTF-16 (2 байта)

    // для символов вне BMP всегда  $ch \in [0xD800...0xDFFF]$ 

}
```

Преобразование из/в UTF-8

// Кодирование string в UTF-8

```
byte[] bytes = Encoding.UTF8.GetBytes(text);
```

// Декодирование из UTF-8 в string

```
string text = Encoding.UTF8.GetString(bytes);
```

// Декодирование содержимого файла из UTF-8

```
string contents = File.ReadAllText(path, Encoding.UTF8);
```

Классы символов

Класс символа	Метод char	Метод Rune
Буква (в любом языке)	char.IsLetter	Rune.IsLetter
Цифра (в любом языке)	char.IsDigit	Rune.IsDigit
Пробел, табуляция, конец строки	char.IsWhiteSpace	Rune.IsWhiteSpace

Конечный автомат для распознавания

Конечный автомат-распознаватель

1. Имеет состояние — перечислимый тип
2. Сканирует слово по одному символу от начала до конца
 - Обработка символа может поменять состояние
 - Обработка символа может вызвать побочные эффекты

Конечный автомат в C#

```
foreach (Rune ch in text.EnumerateRunes()) {  
    switch (state) {  
        case State.SomeState:  
            // ...действие в состоянии SomeState  
            break;  
    }  
}
```

Конечный автомат в C#

```
foreach (Rune ch in text.EnumerateRunes()) {  
    switch (state) {  
        case State.SomeState:  
            // ...действие в состоянии SomeState  
            break;  
    }  
}
```

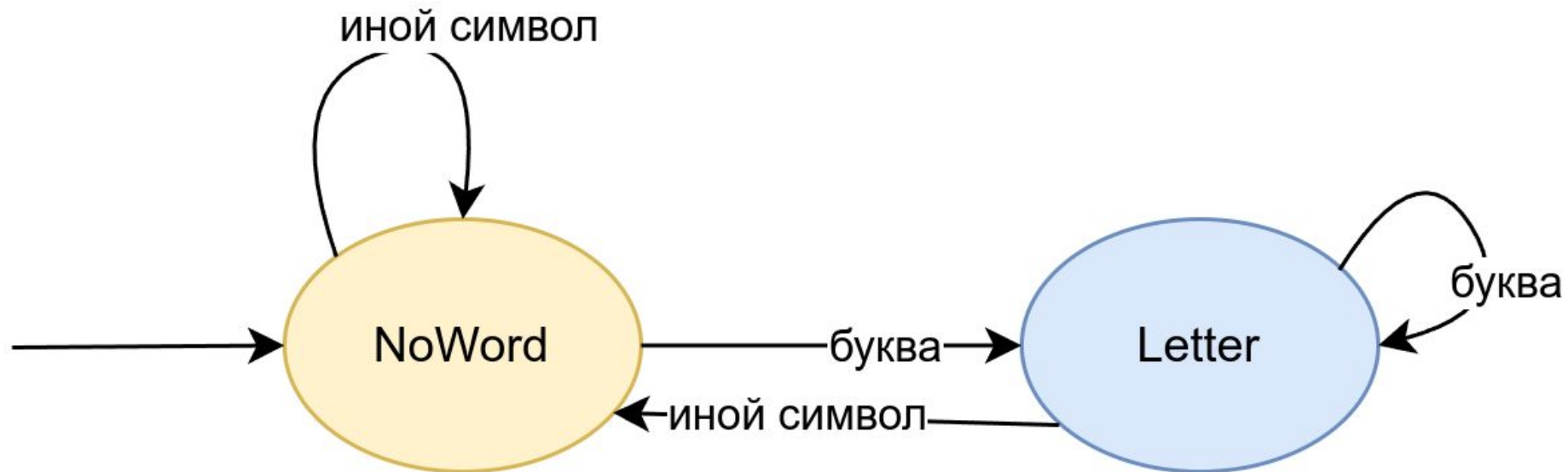


Можно заменить switch на поиск в таблице

Что такое слово?

1. Слово может содержать буквы
 - "Hello"
 - "Привет"

Конечный автомат-распознаватель



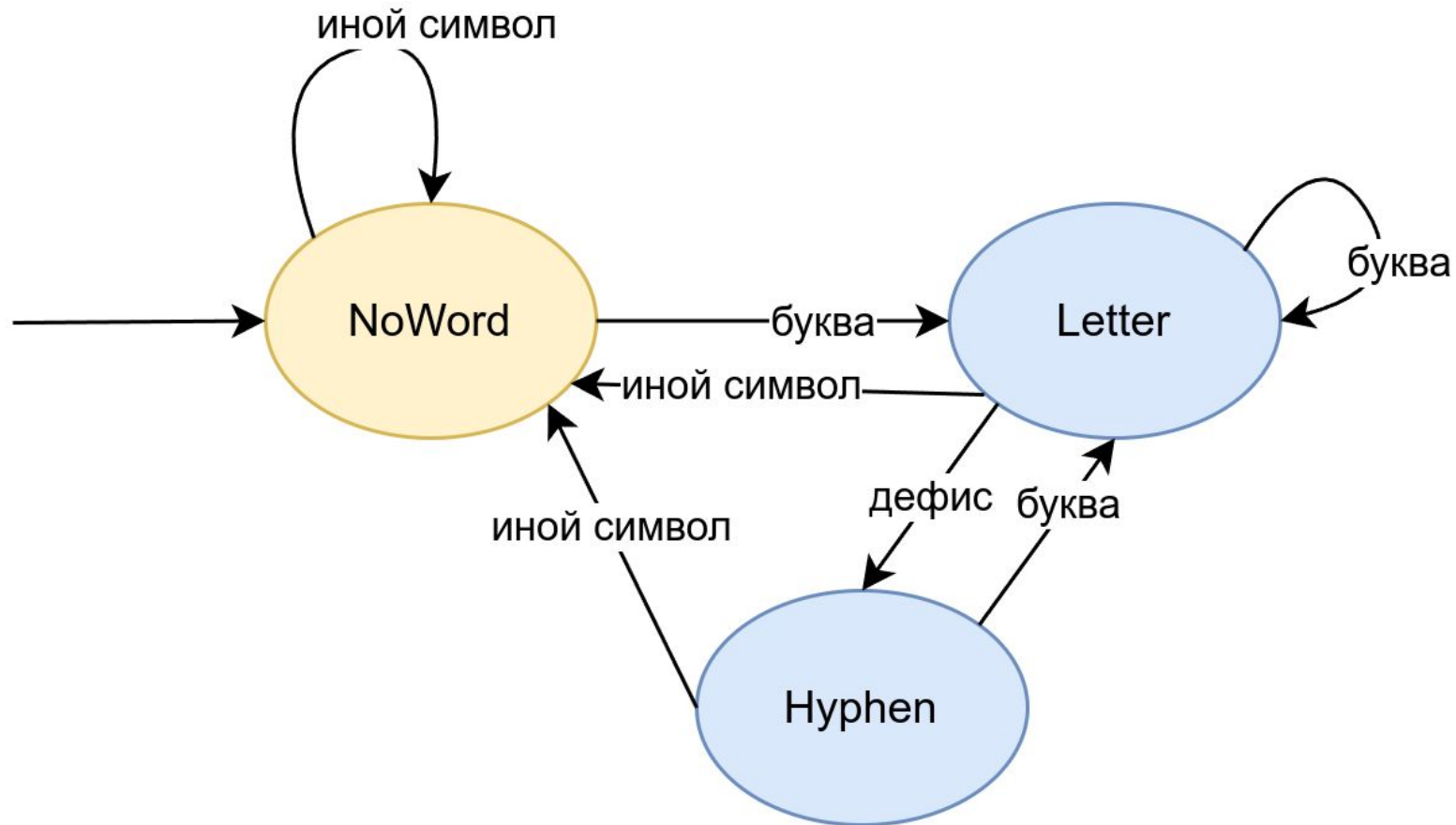
Пример — действие в состоянии NoWord

```
case WordState.NoWord:  
    if (Rune.IsLetter(ch)) {  
        PushCurrentWord();  
        currentWord.Append(ch);  
        state = WordState.Letter;  
    }  
    break;
```

Добавим правило — дефис

1. Слово может содержать буквы
 - "Hello"
 - "Привет"
2. Слово может содержать дефис в середине
 - "что-нибудь"

Конечный автомат-распознаватель



Пример — действие в состоянии Hyphen

```
case WordState.Hyphen:
```

```
    if (Rune.IsLetter(ch)) {
```

```
        currentWord.Append(ch);
```

```
        state = WordState.Letter;
```

```
    } else {
```

```
        // Убираем дефис из конца слова
```

```
        currentWord.Remove(currentWord.Length - 1, 1);
```

```
        state = WordState.NoWord;
```

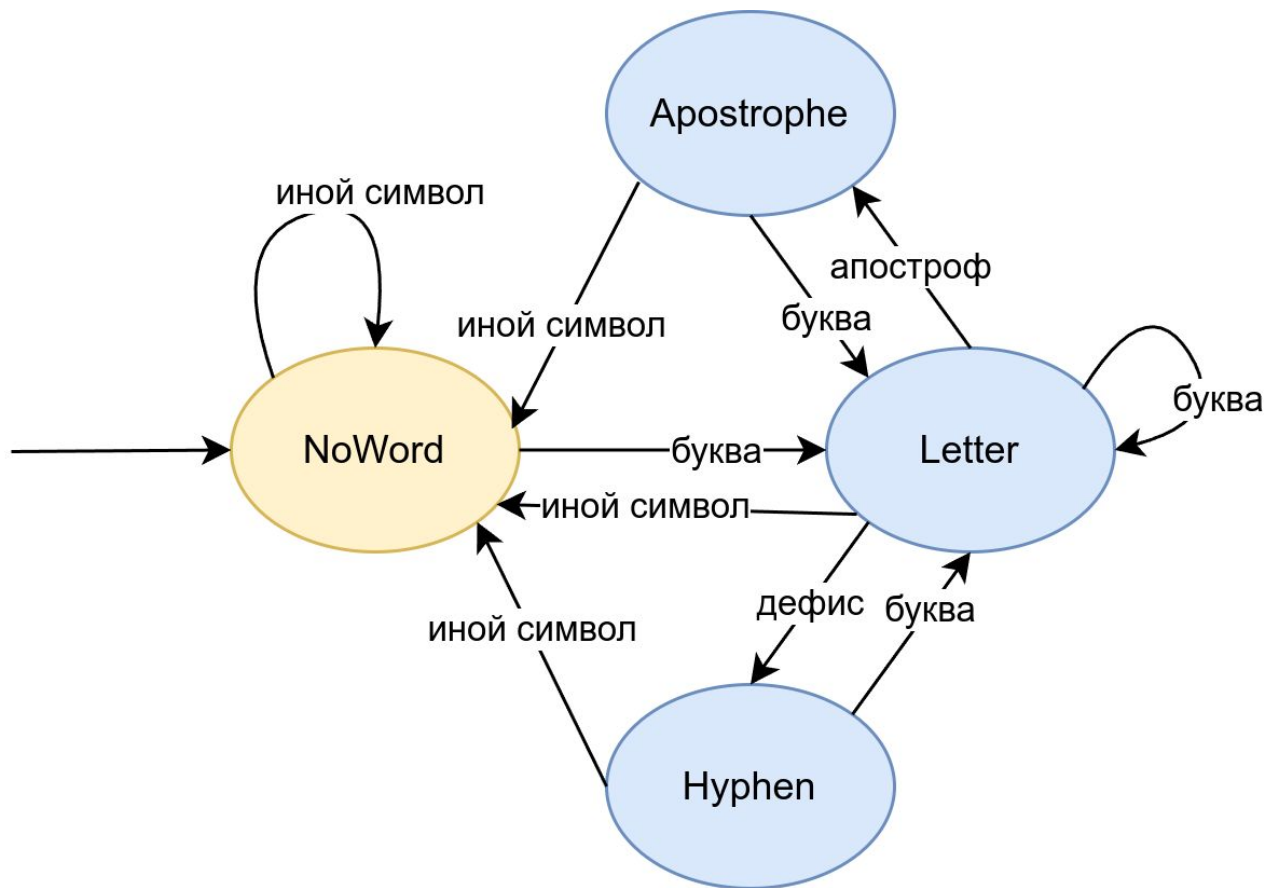
```
    }
```

```
break:
```

Добавим правило — апостроф

1. Слово может содержать буквы
 - "Hello"
 - "Привет"
2. Слово может содержать дефис в середине
 - "что-нибудь"
3. Слово может содержать апостроф в середине или в конце
 - "can't"
 - "cats' toys"

Конечный автомат-распознаватель



Пример — действие в состоянии Apostrophe

```
case WordState.Apostrophe:
    if (Rune.IsLetter(ch)) {
        currentWord.Append(ch);
        state = WordState.Letter;
    } else {
        state = WordState.NoWord;
    }
    break;
```

Конечный автомат и читаемость кода

Функция `TextUtil.ExtractWords`:

- 82 строки кода
- из них 59 строк — внутри `switch`
- реализует конечный автомат — 4 состояния, 8 переходов

См. [Шаблон проекта на C# / .NET](#)

Конец.
Вопросы?