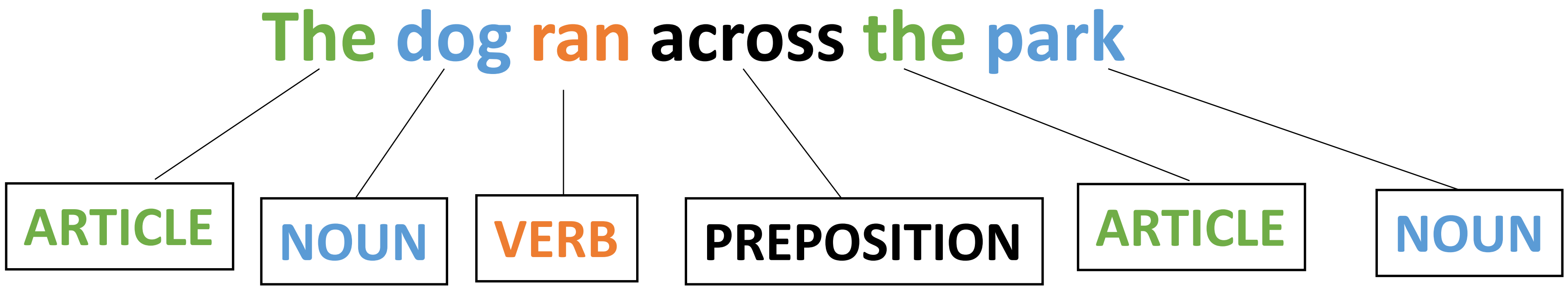
CSE110A: Compilers



The dog ran across the park

ARTICLE NOUN VERB PREPOSITION ARTICLE NOUN

- **Topics**:
  - *Finishing up Scanners*
    - *Token actions*
    - *PLY Scanner*

Finishing up scanner implementations

Scanners we have discussed

- *Naïve Scanner*

- *RE based scanners*
  - Exact match (EM) scanners
  - Start-of-string (SOS) scanners
  - named group (NG) scanners

Which one to use?
Complex decision with performance, expressivity, and token requirements

In practice

- Most scanner generators tend to use SOS semantics
  - You can reason about tokens independently
  - Use fast ”match” implementations under the hood

- Mainstream compilers:
  - have hand coded and hand optimized scanners
  - _very_ fast
  - _very_ hard to modify
  - *Only worth it to do this if you have the need and time*

Moving on

- Token actions
    - Replacement
    - Keywords
    - Error reporting

- Scanner error recovery

Moving on

- **Token actions**
  - Replacement
  - Keywords
  - Error reporting

- Scanner error recovery

First class functions

- A programming language is said to have first class functions if functions can be stored as variables

- Python has great support for this

- Many modern languages have functional programming features, many also help out by doing type checking supporting first class functions.

- In C++
  - Classically: function pointers
  - Newer: supports lambdas
  - Also supports libpcre (RE library)

Functions as part of a token definition

- In our scanners, we give them as the 3rd element in the token tuple definition

- A **token action takes in a lexeme and returns a lexeme**.
    - Possibly the same lexeme

They generally do three things:

- **modify a value**
- **refine a token**
- **modify the scanner state**

Functions as part of a token definition

- Once a token is matched, its **token action** is called on its **lexeme**,

- and the **lexeme it returns is returned from the scanner**,

- **Code example** in the EM

Examples

Token actions generally do three things:

- **modify a value**
- refine a token
- modify the scanner state

Modify a value

- Example using natural language

Modify a value

- PRONOUN = {His, Her, Their}
- NOUN = {Dog, Cat, Car, Park}
- VERB = {Slept, Ate, Ran}
- ADJECTIVE = {Purple, Spotted, Old}

Tokens

Tokens Definitions

Modify a value

Tokens		Tokens Definitions
• PRONOUN	=	{His, Her, Their}
• NOUN	=	{Dog, Cat, Car, Park}
• VERB	=	{Slept, Ate, Ran}
• ADJECTIVE	=	{Purple, Spotted, Old}

Example:
Can change any pronoun value to gender neutral (“Their”)

Modify a value

- Example using types

Some ML frameworks experiment with lower precision, e.g., **`float16`**

Change code to use lower precision

```
float x, y;
return x+y;
```

The scanner can be made To easily change float to float16 with a token action

```
float16 x, y;
return x+y;
```

Examples

Token actions generally do three things:

- modify a value
- **refine a token**
- modify the scanner state

Keywords: *(a better way to tokenize!)*

Keywords

TOKENS

```
ID     = [a-z]+
NUM    = [0-9]+
ASSIGN = ”=“
PLUS   = “+”
MULT   = “*”
IGNORE = [“ “, “\n”]
```

KEYWORDS

```
[(INT,“int”), (FLOAT, “float”) ...]
```

Keywords

```
TOKENS
ID     = [a-z]+
NUM    = [0-9]+
ASSIGN = ”=“
PLUS   = “+”
MULT   = “*”
IGNORE = [“ “, “\n”]

KEYWORDS
[(INT,“int”), (FLOAT, “float”) ...]
```

Code example in EM Scanner

Examples

Token actions generally do three things:

- modify a value
- refine a token
- **modify the scanner state**

Modifying State

A major use case is for error reporting

- Line number
- Column number

Doesn’t fit well in the framework of our homework

- Our homework has scanners importing a token definition file.
- As configured it is not clear how to proceed!!

- *Maybe some of you can think of a design where this works with our homework*

NEXT:

We will start on Module 2 on parsing!