

Eric Wanner-Garnier

CSCI 476 Spring 23

Capstone Document

26 April 2023

Section 1: Program

Accompanying zip file: source.zip

Section 2: Teamwork

Team Member 2 wrote the documentation for the compiler as well as a high level test suite for the Catscript features and Team Member 1 wrote the source code to the grammar, features, and tests.

Section 3: Design pattern

We used memoization in `src/main/java/edu/montana/csci/csci468/parser/CatscriptType.java` so that once a list type was checked for the first time, we wouldn't have to reinitialize a `CatscriptType` object each time the `getListType` method was called, and instead the initial `CatscriptType` object is stored in a hash map and reused.

Section 4: Technical writing

Introduction

Catscript is a simple scripting language. Here is an example:

```
var x = "foo"
print(x)
```

Features

Catscript types

Catscript is statically typed using the following types:

int - a 32 bit integer

string - a java-style string

bool - a boolean value

list - a list of values of a given type

null - the null type

object - any type of value

Statements

For loop:

- General syntax

```
'for', '(', IDENTIFIER, 'in', expression ')', '{', { statement }, '};'
```

- Example

```
var array = ["a","b","c"]
var numElements = 0
for(x in array){
    print(x)
    numElements = numElements + 1
}
print("There were " + numElements + " elements in the array")
```

If statment:

- General syntax

```
'if', '(', expression, ')', '{', { statement }, '}' [ 'else', ( if_statement | '{', { statement }, '}' ) ];
```

- Example

```
var a = 1
var b = 5

if(a > b){
    print("a is greater than b")
} else {
    print("a is less than b")
}
```

Print statement:

- General syntax

```
'print', '(', expression, ')'
```

- Example

```
print("Hello from CatScript!")
```

Variable statment:

- General syntax

```
'var', IDENTIFIER, [':', type_expression, ] '=', expression;
```

- Example

```
var a = 1
var b:int = 4
print(a + " and " + b + " are both of type int")
```

Assignment statement:

- General syntax

```
IDENTIFIER, '=', expression;
```

- Example

```
var a = 3
var b = a*2
print("a is " + a + " and b is " + b)
```

Function declaration statement:

- General syntax

```
'function', IDENTIFIER, '(', parameter_list, ')' + [ ':' + type_expression ], '{', { function_body_statement }, '');
```

- Example

```
function add(val1:int, val2:int){
    var result = val1 + val2
    return result
}

print("one plus two equals ")
print(add(1,2))
```

Function call statment:

- General syntax

```
IDENTIFIER, '(', argument_list , ')'
```

- Example

```
function add(val1:int, val2:int){
    var result = val1 + val2
    return result
}

print("one plus two equals ")
print(add(1,2))
```

Function return statement:

- General syntax

```
'return' [, expression];
```

- Example

```
function add(val1:int, val2:int){  
    var result = val1 + val2  
    return result  
}
```

```
print("one plus two equals ")  
print(add(1,2))
```

Expressions

Equality expression:

- General syntax

```
comparison_expression { ("!=" | "==") comparison_expression };
```

- Example

```
print(5 != 2)
```

Comparison expression:

- General syntax

```
additive_expression { (">" | ">=" | "<" | "<=" ) additive_expression };
```

- Example

```
print(5 >= 2)
```

Additive expression:

- General syntax

```
factor_expression { ("+" | "-" ) factor_expression };
```

- Example

```
print(5 - 1)
```

Factor expression:

- General syntax

```
unary_expression { ("/" | "*" ) unary_expression };
```

- Example

```
print(12 / 3)
```

Unary expression:

- General syntax

```
( "not" | "-" ) unary_expression | primary_expression;
```

- Example

```
print(not true)
```

Primary expression:

- General syntax

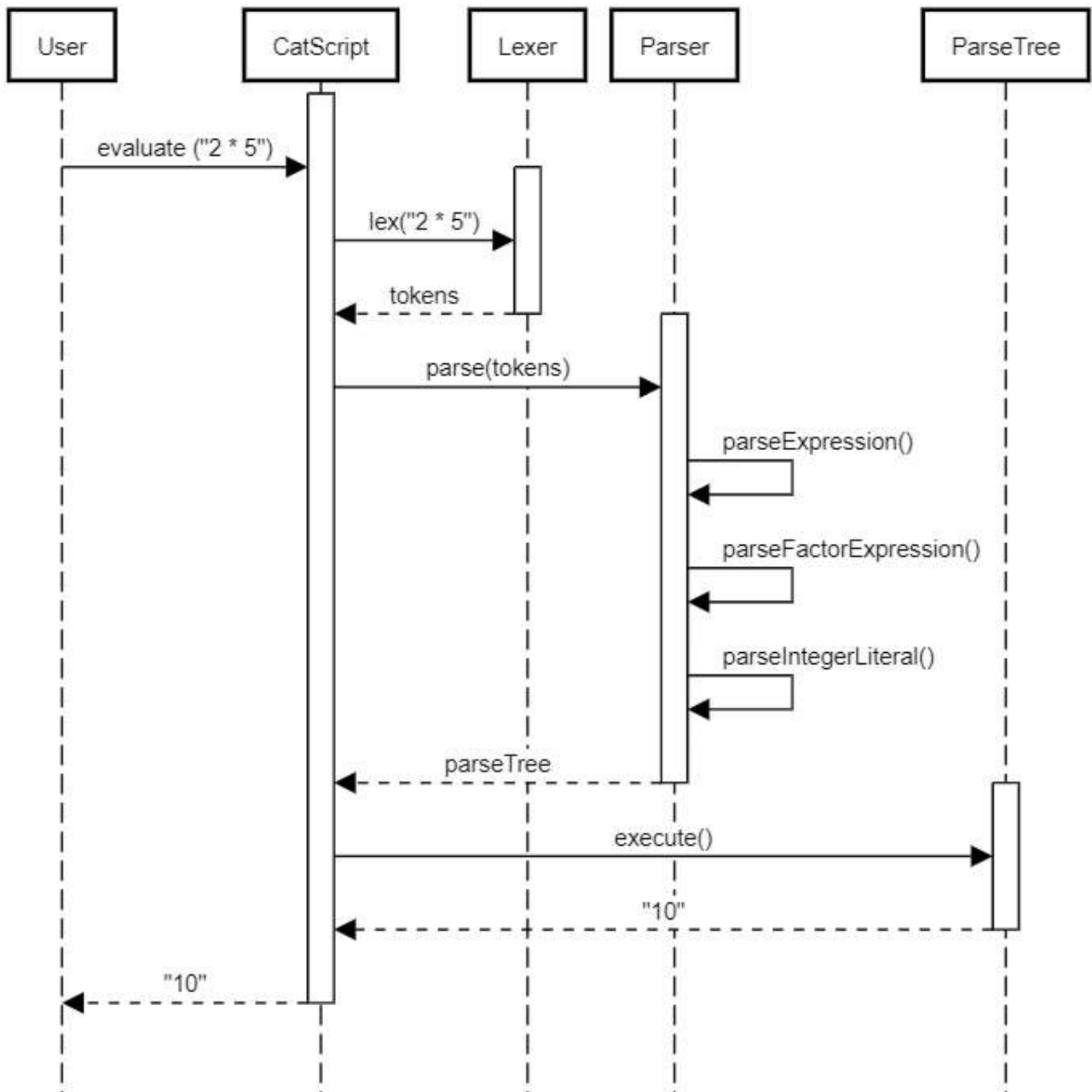
```
IDENTIFIER | STRING | INTEGER | "true" | "false" | "null" | list_literal | function_call | "(", expression, ")"
```

- Example

```
print("This is a primary expression!")
```

Section 5: UML diagram

Catscript Multiplication Sequence Diagram



Section 6: Design trade-offs

We decided to design our program with recursive descent instead of the alternative of taking the specified grammar and using a parser generator. While a parser generator is quick, we did not choose this option since the generated code can be both bloated and complicated to debug or modify.

Section 7: Software development life cycle model

We used a test driven development model during development. Starting the project we had a test suite that specified the Catscript grammar and functionality. The main downfall I found with this approach is that I did not consider all the edge cases all of the time, instead considering the tests completion as complete functionality. However, it did help to give insight to the intended functionality and cohesion from our team's design.