

Makayla Broyles
CSCI 468
Spring 2023
Team: Cameron Wilcox

Program Specifications

```
catscript_program = { program_statement };

program_statement = statement |
                    function_declaration;

statement = for_statement |
            if_statement |
            print_statement |
            variable_statement |
            assignment_statement |
            function_call_statement;

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

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

print_statement = 'print', '(', expression, ')'

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

function_call_statement = function_call;

assignment_statement = IDENTIFIER, '=', expression;

function_declaration = 'function', IDENTIFIER, '(', parameter_list, ')' +
                      [ ':' + type_expression ], '{', { function_body_state
ment }, '}' ;

function_body_statement = statement |
                        return_statement;

parameter_list = [ parameter, {',', parameter } ];

parameter = IDENTIFIER [ , ':', type_expression ];

return_statement = 'return' [, expression];

expression = equality_expression;

equality_expression = comparison_expression { ("!=" | "==") comparison_expres
sion };
```

```

comparison_expression = additive_expression { (">" | ">=" | "<" | "<=" ) addi
tive_expression };

additive_expression = factor_expression { ("+" | "-" ) factor_expression };

factor_expression = unary_expression { ("/" | "*" ) unary_expression };

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

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

list_literal = '[', expression, { ',', expression } ']';

function_call = IDENTIFIER, '(', argument_list , ')'

argument_list = [ expression , { ',', expression } ]

type_expression = 'int' | 'string' | 'bool' | 'object' | 'list' [, '<' , type
_expression, '>']

```

Teamwork

For our team we had a main developer for each project and a quality/business analyst. To do this, both of us completed the project as a developer we then swapped tests and documentation. This verified that we both learned the material but also worked collaboratively. Occasionally, we also met up to help debug each other's code if one of us was needing some help.

Design Pattern

The design pattern we used is Memoization which is a type of caching. It allows for a function to only run once and stores the result in memory. Therefore, when it is run again, it returns the cached result instead of running the logic.

```
public static CatscriptType getListType(CatscriptType type) {  
    CatscriptType listType = LIST_TYPES.get(type);  
    if (listType == null) {  
        listType = new ListType(type);  
        LIST_TYPES.put(type, listType);  
    }  
    return listType;  
}
```

o

Technical Documentation

Catscript Guide

Introduction

Catscript is a simple scripting language written for CSCI 468 (Compilers). Here is an example:

```
var x = "foo"  
print(x)  
function bar(y,z):int {  
    return y+z  
}
```

Features

For Loops

A for loop is used to iterate over a block of code a set number of iterations, typically iterating over a list of items. In Catscript, for loops require a list of objects to iterate over. Here is an example:

```
for (x in [1, 2, 3]) {  
    print(x)  
}
```

Print Statement

A print statement is used to send output to the terminal. Here is an example:

```
print("Hello World!")
```

If Statement

An if statement is used to branch depending on certain criteria. It may be followed by an **else** if there is code that needs to be executed if the boolean logic is not true. Here is an example of both:

```
var x = 1  
if(x != 1) {  
    print("not 1")  
} else {  
    print("is 1")  
}
```

Functions

Functions are used to extract blocks of code that may have a special significance. You may need to give it inputs, in which case parameters may need to be defined. A function may be written to do a special computation. If that is the case a **return** statement may be used as the last

statement in a function to return the value. Here is an example of two functions, one which has no parameters and returns nothing, and one which computes the square of a number:

```
function print_hello() {  
    print("hello")  
}  
  
function square(x) : int {  
    return x*x  
}
```

Types and Operators

Types

The Catscript language supports six different types:

- int - a 32 bit integer
- string - a java-style string
- bool - a boolean value
- list - a list of value with the type 'x'
- null - the null type
- object - any type value

These can be seen in the examples below:

```
var an_integer = 100  
var a_string = "hello"  
var a_bool = true  
var a_list = [1, 2, 3]  
var another_list<bool> = [true, false, true]  
var null_type  
var any_type : object = "some_type"
```

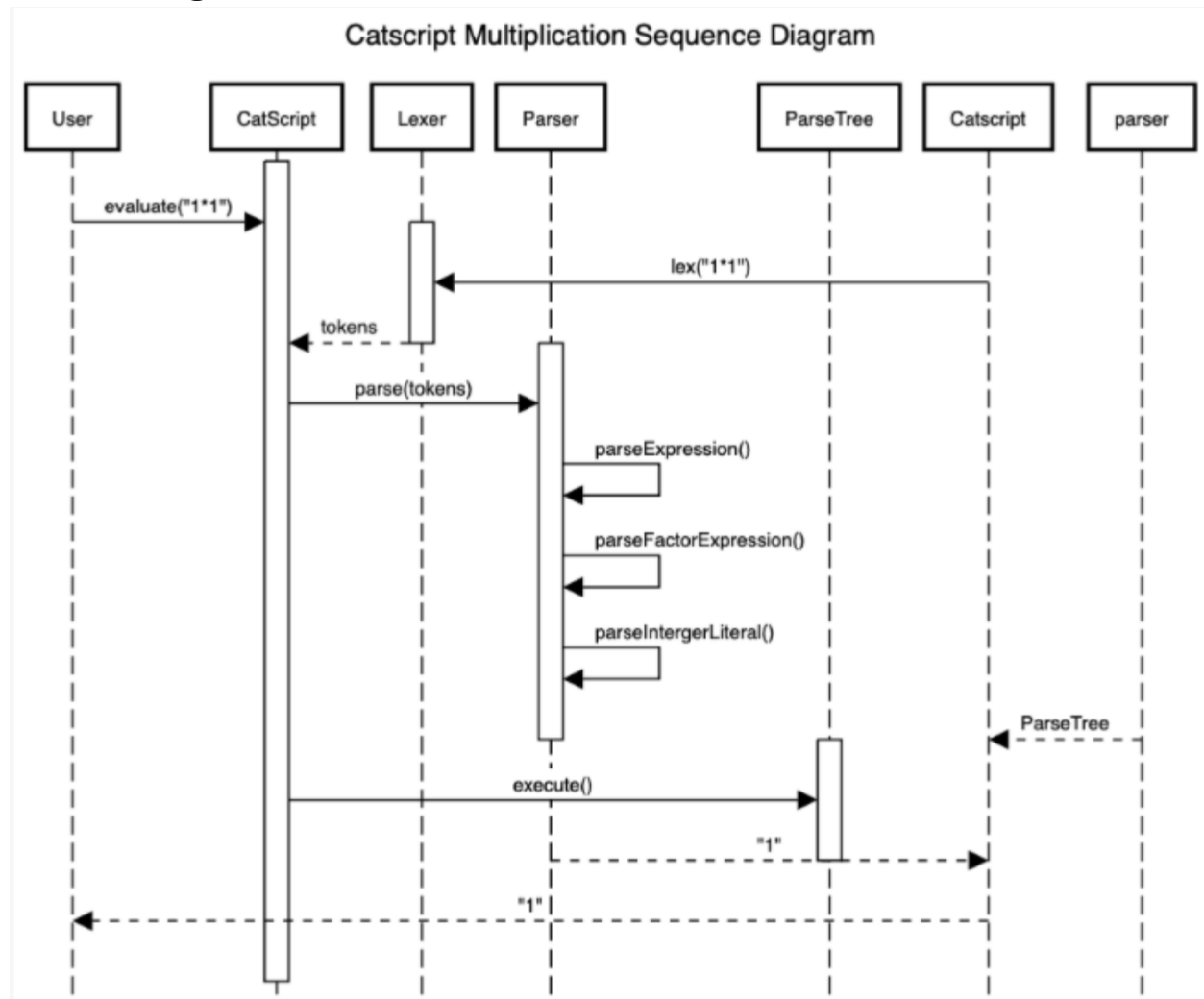
As shown in the last variable declaration, you can declare a variable type explicitly. In this example, lists are declared

Operators

The Catscript language supports many different types of operations on these types. These include:

- != (not equals)
- == (equals equals)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)
- + (addition and string concatenation)
- - (subtraction and negative)
- / (division)
- * (multiplication)
- not (not operator)

UML Diagram



Design Trade-offs

The main design trade off when writing the Catscript parser was using recursive decent instead of a parser creator. Parser creators are normally encouraged when learning how to create a parser but using recursive decent allows you to learn to program the parser correctly. Parser creators create code that is hard to understand and debug which is not always the best if you are still learning. Using recursive decent allowed me to continue to improve my programming skills and really understand how a compiler works.

Software Development Life Cycle

For my capstone project I used test driven development. This is where all requirements are translated into tests and as you develop you continue to run the tests against the classes until they all pass. This model made development simple since there was only one main developer. This development life cycle is one of my favorites because it really enforces the true requirements as you develop.