CSCI 468: Compilers
Senior Team Portfolio

Devon Salveson & Samuel Ertischek

Spring 2023

Part 1: Program

Program Zip
See the source.zip in this directory

Part 2: Teamwork

We both did equal amounts of work on our capstone project. We split the tasks equally between us and frequently collaborated with each other on tasks.

Part 3: Design Pattern

The design pattern that we used in the project was memoization. Memoization is used for optimization to make the program more efficient and thus faster. This is done by storing results in the cache and pulling that same result from the cache when it is next needed instead of computing it again.

Part 4: Technical Writing

Introduction

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

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

CatScript Types

CatScript is statically typed, with a small type system as follows

- 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 of value

Features

For Loops

For Loops will go through every variable in a list and use that variable for the function it has.

Here is an example:

```
list<int> x = [1,2,3,4,5]
for(y in x){print(y)}
```

The code above would output 1 2 3 4 5. This would be because the for statement would go through each variable in the list x and print out that variable.

If Statements

An if statement will run the code it has if the expression it uses is true. If there is an else statement below the if statement it will run the code it has if the if statement is false, otherwise it will continue through the program

Here is an example:

```
var x = 5
if(x == 5){print("x is 5")}
Else{print("x is not 5")}
```

The code above would output x is 5 since the variable x is equal to 5. If x was not equal to 5 the code would output x is not 5.

Print Statement

A print statement will print out whatever it contains whether that is a variable, expression or a string

Here is an example:

```
print("Hello World")
var x = 5
print(x)
```

The code above would output Hello World and 5.

Assignment Statements

In catscript you can change the value of a variable using an assignment statement

Here is an example:

```
var x = 5
print(x)
```

```
x = 10
print(x)
```

The code above would output 5 and then output 10 since the variable is first 5 and it is then changed with an assignment statement to 10.

Expressions

There are many different things that you can do with variables in catscript. The things that you can do is listed below

Equality:

- **!=**
    - Compare the variable to something else to see if the variable is not equal to what it is being compared to
- **==**
    - Compare the variable to something else to see if the variable is equal to what it is being compared to

Comparison:

- **>**
    - Compare the variable to something else to see if the variable is greater than what it is being compared to
- **>=**
    - Compare the variable to something else to see if the variable is greater than or equal to what it is being compared to
- **<**
    - Compare the variable to something else to see if the variable is less than what it is being compared to
- **<=**
    - Compare the variable to something else to see if the variable is less than or equal to what it is being compared to

Additive:

- **+**
    - Add the two objects together
- **-**
    - Subtract the second object from the first object

Factor:

- **/**
    - Divide the first object by the second object
- *****
    - Multiply the two objects together.

Functions

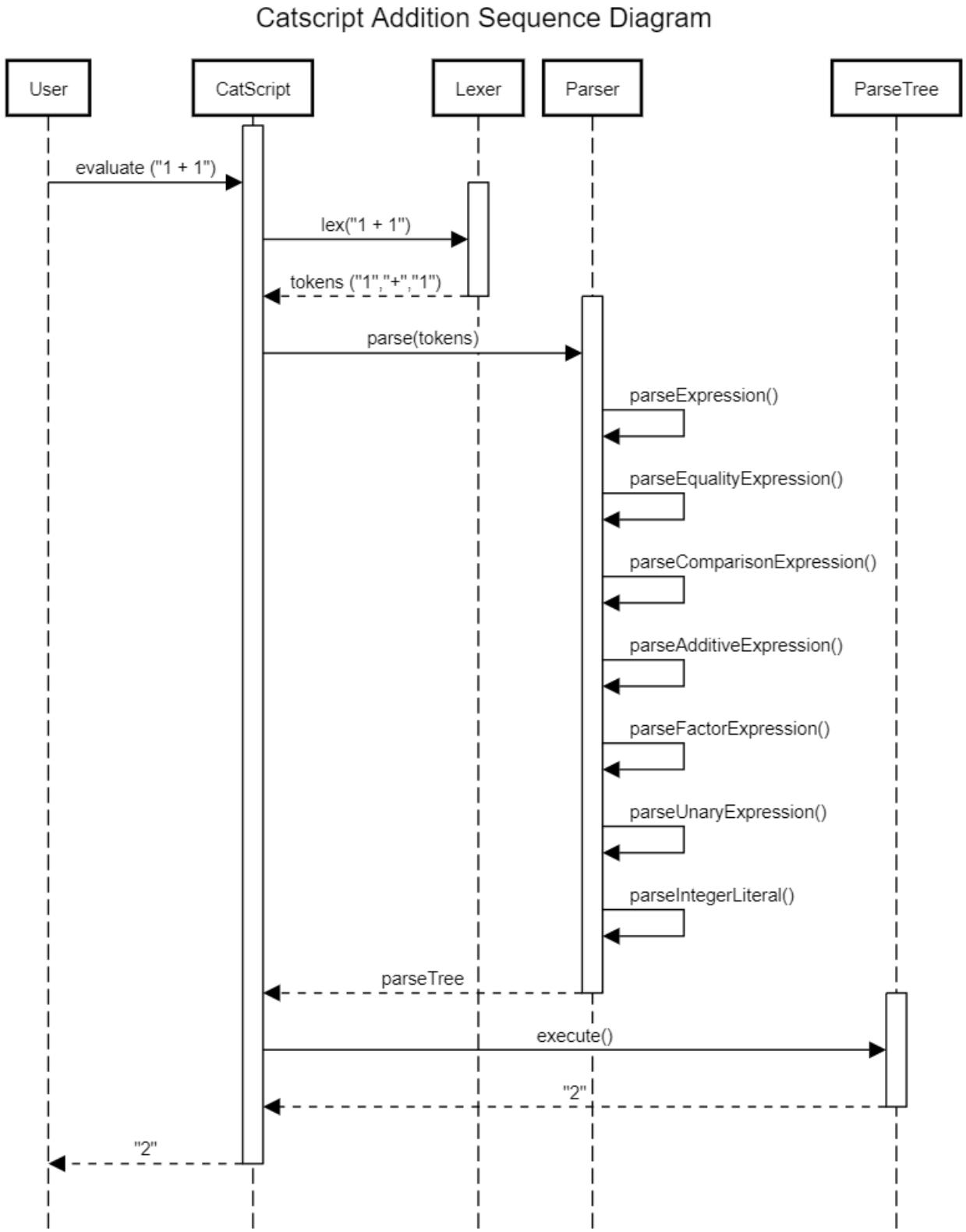
A Function is a chunk of code that you can call to use repeatedly. An example is listed below

```
Function printnum(var x)
{
      print(x)
}

printnum(5)
printnum(10)
printnum(15)
```

The code above would call the function printnum and output 5 10 15.

Part 5: UML


Catscript Addition Sequence Diagram
User
CatScript
Lexer
Parser
ParseTree
evaluate ("1 + 1")
lex("1 + 1")
tokens ("1","+","1")
parse(tokens)
parseExpression()
parseEqualityExpression()
parseComparisonExpression()
parseAdditiveExpression()
parseFactorExpression()
parseUnaryExpression()
parseIntegerLiteral()
parseTree
execute()
"2"
"2"

Part 6: Design Trade-offs

The design trade off that we had in our project was the decision to use the recursive descent algorithm for creating parsers instead of a parser generator. Parser generators are programs that take a language specification and generates a parser for that specification. Whereas recursive descent is a parser that uses recursion to process the input without backtracking. We decided to use the recursive descent model due to it being easier to understand, it letting us understand the recursive nature of grammars and the fact that recursive descent is used way more in the industry.

Part 7: Software development life cycle model

For our Software development life cycle model we used Test Driven Development. This means that tests were created for each functionality before hand and if our code fails the test then new code is written until it passes the test. This was very useful for our project because it allows us to avoid duplication of code and minimize the amount of code needed to check functionality. It also ensures that we are meeting the requirements needed for out project.