CSCI Capstone

Terris Dietz

CSCI 468

Spring 2024

Section 1: Program

Compiler Program attached in included zip file TerrisDietzCapstone.zip

Section 2: Teamwork

Utilized Teamwork by working with a Teammate to create dynamic testing cases to test the final code. Provided code is present in the Zip File and will also be presented below.

```
public class partnerTest extends CatscriptTestBase {

    @Test
    public void ifElseInsideForWorks() {
        assertEquals("0\n1\n0\n", executeProgram("for (y in [1,2,3]){\n" +
                "if (y  == 2){\n" +
                "print(1)\n" +
                "}else{\n" +
                "print(0)\n" +
                "}}"
        ));
    }

    @Test
    public void functionCallInsidePrintWorks() {
        assertEquals("100\n", executeProgram("function foo() : int {" +
                "var x = 100" +
                "return x" +
                "}" +
                "print(foo())"
        ));
    }

    @Test
    public void nestedForLoopsWork() {
        assertEquals("2\n3\n4\n3\n4\n5\n", executeProgram("for (x in [1,
2]){" +
                "for (y in [1, 2, 3]){" +
                "print (x + y)" +
                "}}"
        ));
    }
```

Teamwork was also used to generate documentation for the Catscript programming language. Documentation is found in Catscript.md file in project folder. Will also include it as an appendix.

Section 3: Design pattern

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

In the final project we used the Design Pattern Memoization to optimize the getListType() function. Memoization improved timing by Caching redundant calls with the same parameters. This improves efficiency by minimizing redundant computation.

Section 4: Technical writing

~~~~markdown
# Catscript Guide

This document should be used to create a guide for catscript, to satisfy capstone
requirement 4

## Introduction

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

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

## Features

Types:
    Catscript provides the following types:
        Int
        String
        List
        Bool
        Object
        Null

Program Structure
~~~~

A Catscript program consists of a collection of statements, which are either function declarations or program statements. Catscript provides the following program statements:

- if statement
- for statement
- print statement
- variable declaration statement
- assignment statement
- function call statement
- return statement

Statements

If statement

The if statement allows you to execute a block of statements based on whether or not the boolean statement provided to it is true. An if statement is structured as follows:

```
if(expression) {
    //statements
}
```

If statements can also use an else to execute a different block of statements if the boolean statement provides evaluates to false. An if-else statement is structured as follows:

```
if(expression){
    //statements
}else{
    //more statements
}
```

For statement

The for statement allows you to loop over a block of statements using a range of values. The for statement is structured as follows:

```
for (identifier in expression){
    //statements
}
```

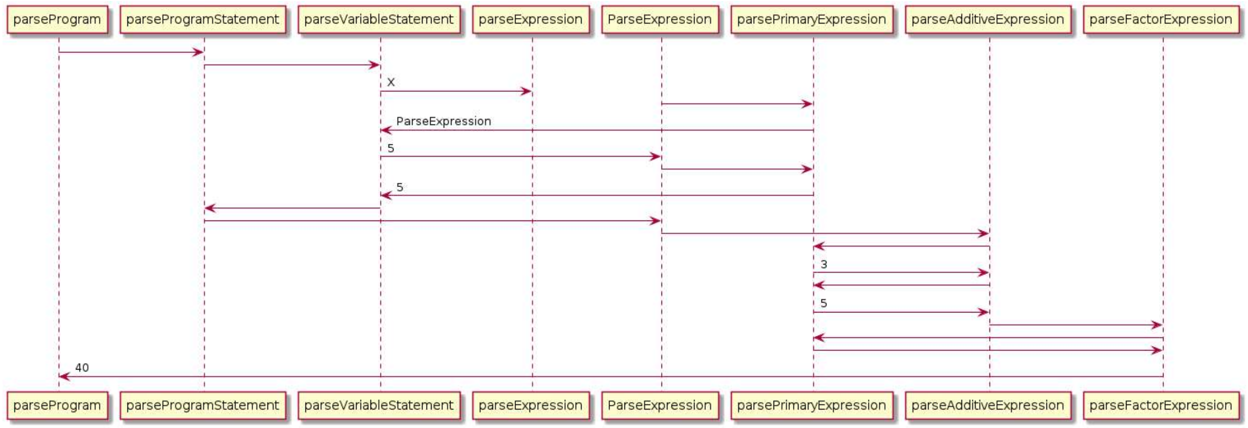
Print statement

The print statement allows you to output a value to the console. A print statement is structured as follows:

```
    print(expression)

Variable declaration statement
    The variable declaration statement allows you to create an assign a value to 
a new variable. The variable
    declaration statement is structured as follows:

    Var identifier [type_expression] = expression

Assignment statement
    The assignment statement allows you to reassign a new value to an existing 
variable. The assignment statement is
    structured as follows:

    identifier = expression

Function call statement
    The function call statement allows you to call a function. The function call 
statement is structured as follows:

    function_call

Return statement
    The return statement is used to return a value and end the execution of a 
function. The return statement is structured as follows

    return(expression)

#### Expressions

Additive Expression
    The additive expression allows for the addition or subtraction of strings, 
integers, or objects using the '+' and '-' operators.

        Statement: 1 + 2
        Output: 3

        Statement: 2 - 1
        Output: 1

        Statement: 'cap' + 'stone'
        Output: 'capstone'


Comparison Expression
```

```
    The comparison expression allows for the comparison of two integer values
using the '>', '<', '>=', and '<=' values
    to evaluate to a boolean true or false.

        Statement: 1 > 2
        Output: false

        Statement: 2 > 1
        Output: true

        Statement: 2 >= 2
        Output: true

Equality Expression
    The equality expression allows for checking the equality or inequality
between two values using the '==' and '!='
    operators, resulting in a boolean true or false.

        Statement: 1 == 2
        Output: false

        Statement: 2 == 2
        Output: true

Factor Expression
    The factor expression allows for multiplication or division operations on
integers or numeric values using
    the '*' and '/' operators.

        Statement: 3 / 6
        Output: 2

        Statement: 4769294781 * 2
        Output: 9538589562

Unary Expression
    The unary expression allows for applying negation to a single operand, using
'-' or 'not' respectively.

        Statement: -5
        Output: -5

        Statement: not true
        Output: false
```

Section 5: UML



Above is the UML Sequence Diagram for the below code:

var x = 5

(3 + 5) * x

Section 6: Design trade-offs

In this project we used the recursive decent algorithm to create a parser for the Catscript programming language. In industry using parsing generators is another option for creating a parser. In this class I appreciate the operturnity to learn how to build one from scratch but would like to take a second to look at the benefits we are missing by using this route.

One benefit of using a parser generator is the ease of growing the language and the automation of the generation. This would have been more time efficient in this class and would of made the project much more scalable if we wanted to add to the language in the future. With the method we used if we wanted to add additional functionality to the language we would have to build a significant amount of code to handle the changes. If the parser generator was used we could define the language of the change and the code would be made for us.

One benefit of creating our own parser that I see as a big trade off in its favor is the ease of debugging. Since I was able to understand and read the code in the back end I was able to troubleshoot issues as they arrived. In the parser Generator the code would have been harder to parse so trouble shooting would have been more time consuming process

Overall both have pros and cons but for a project of this size I feel like the benefits of learning the algorithm outweighed the convenience and scaleablility of a generator.

Section 7: Software development life cycle model

Four our development we utilized Test-Driven Development (TTD). TDD is utilized by creating a series of tests before the code is developed that the generated code is ran against to verify operation. These tests are often small and test small parts of the program so as you are developing you can see if small pieces work instead of waiting till the end.

Overall, I found this to be a great way to build this project as the project can be split into very distinct parts, Tokenizing, Parsing, Eval, and Code Generation. If did not have a way to test these operations as we went along finding bugs near the end would be overwhelming.

Also, with testing you are able to use the debugger more efficiently since you have tests you can iterate through and see when the issues happen in small controlled functions. Overall if I was in a situation again where I had the opportunity to develop with this structure I would do it as it is rewarding throughout the process cause when you get a test to pass it is a tangible reward instead of waiting to the end of the process.