

Catscript Tokenizer

Csci468

Spring 2022

Brian Keith

Armand LaPlume

Section 1: Program

I have included a zip file of the complete project

Section 2: Teamwork

Teammate 1 was the primary coder, they wrote most of the code for the project and satisfied all the given tests, they made up about 95% of the time spent on this project. Teammate two was the tester, They provided tests that ensured the project met the requirements. Teammate 2 also provided the documentation, they made up about 5% of the time spent on this project

Section 3: Design pattern

In this project we used the Flyweel design pattern, sometimes called memoization. It is used to increase efficiency by reusing objects instead of creating new ones. A great example of this design being used in the code is in the getListType method of CatscriptType. located specifically at lines 37-45 in CatscriptType.java, the code snippet is included below.

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

Section 4: Technical writing

The following is the technical document for the Catscript Project

Introduction

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

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

Features

Expressions

Expressions in Catscript always evaluate to a value. They follow an order of precedence of unary, factor, additive, comparison, and equality where unary expressions are the first ones evaluated and equality expressions are the last ones evaluated. There are also primary expressions which are the lowest level expressions and each one is either an identifier, string, integer, boolean true or false, null, list, function call, or it can be a parenthesized expression which will then recursively parse another expression.

Unary Expression

The unary expression is used to invert a boolean or an integer. A unary expression is the word “not” or the “-” symbol followed by a boolean or an integer respectively. Here is an example where the result printed is -5:

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

Factor Expression

The factor expression is a multiplicative expression. It’s syntax is an integer followed by either a “*” or “/” followed by another integer. Here is an example where the result of x would be 28:

```
var x = 4 * 7
```

Additive Expression

An additive expression can either be a mathematical expression where to integers are being added or subtracted or it can be string concatenation. An additive expression is an expression followed by either a “+” or “-” followed by another expression. Here are two examples where x would be the integer 14 and y would be the string “Catscript”:

```
var x = 8 + 6  
  
var y = “Cat” + “script”
```

Comparison Expression

A comparison expression evaluates to true if one integer is greater than, greater than or equal to, less than, or less than or equal to another integer, depending on which operator is used. A comparison expression is an expression followed by a ">", ">=", "<", or "<=" followed by another expression. Here are two examples where x is true, and y is false:

```
var x = 4 >= 1
```

```
var y = 7 < 4
```

Equality Expression

An equality expression compares two objects to see if they are equal or not equal depending on the operator used. An equality expression is an object followed by a "==" or "!=" followed by another object. Here are two examples that are both true.

```
var x = 1 == 1
```

```
var y = true != null
```

Statements

Print Statement

This prints the contents of the print statement exactly like print statements in other languages. A print statement is the keyword print followed by a "(" followed by the expression to be printed followed by a ")". Here is an example where 5 is printed:

```
Var x = 5
```

```
Print(x)
```

Variable Statement

A variable statement creates a new variable and assigns it a value. A variable can have its type explicitly defined or it can infer it from its assigned value. A variable statement is the var keyword followed by the name of the variable optionally followed by a ":" and a type followed by a "=" followed by an expression. Here are some examples of var statements:

```
var x = 10
```

```
var y = true
```

```
var z = "Catscript"
```

Assignment Statement

An assignment statement assigns a new value to a variable. An assignment statement is the name of the variable followed by a "=" followed by the new value. Here is an example where the value 1 would be printed:

```
var x = 2
```

```
x = 1
```

```
print(x)
```

For Statement

A for statement in Catscript is like a for statement in any other language. It executes a set of statements a certain number of times. A for statement the keyword for followed by a "(" followed by the name of a variable local to the for loop (typically i) followed by the keyword in followed by a list followed by a ")" and "{" followed by any number of body statements followed by a closing "}". Here is an example that would print 1, 2, 3:

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

If Statement

An if statement is used to execute a block of code if the expression in it evaluates to true. An if statement is the keyword if followed by a "(" followed by an expression followed by a ")" and "{" followed by the statements to be executed followed by a closing "}". An if statement can optionally be followed by an else if or else statement to either check if another expression evaluates to true and execute a list of statements or to unconditionally execute a list of statements if the expression in the if statement did not evaluate to true. Here is an example where "if statement" is printed:

```
var x = 5
```

```
if (x == 5){
```

```
  print("if statement")
```

```
} else { print("else statement") }
```

Function Definition Statement

A function definition statement in Catscript defines a function with a given number of parameters which each can optionally be assigned a type, an optional return type, and a body of statements to be executed. Here are some example definitions:

```
function foo(x) { print(x) }
```

```
function foo1() : int {
```

```
  var x = 42
```

```
  return x
```

```
}
```

```
function foo2(x : int) : int {
```

```
  return x + 1
```

Section 5: UML.

A Class diagram for the class ParseElement and its related classes is shown below



Section 6: Design trade-offs

For this project one of the biggest design tradeoffs was Using a recursive descent structure instead of a parser generator. This decision was made because writing a recursive descent parser gives a much better understanding of what is actually happening as opposed to a parser generator handling which does most of the more technical stuff automatically, leaving the programmer with larger chunks of generated code that they did not write and may not be able to understand. The tradeoff is that using recursive descent does require the programmer to write more code by hand, and parser generators tend to be more standard in a university setting.

Another design decision was not using a visitor pattern, everything was done directly in the relevant Class. Not using the visitor pattern creates a simpler end product that is more streamlined to write, however, it would be harder to maintain for a growing system. because the Catscript project was a relatively small project it made sense to not use the pattern in exchange for simplicity and ease of coding and debugging.

Section 7: Software development life cycle model

This project was guided by test-driven development. The primary coder was given an extensive test suite and the goal was to get the project to pass all the tests. This was accomplished using progressive development of building blocks, first tokenizing, then parsing, then finally compiling.