COMPILERS CSCI-468

Spring 2022

Mason Medina
Tester: Rory McLean
Gianforte College of Computer Science
Montana State University

Program

The Source code can be found at PUT LINK HERE

Teamwork

The work done on this project was broken into several sprints of test driven implementation. The different sprints of testing were broken up as follows; Tokenization, Parsing, Evaluation, Bytecode generation, and Partner driven testing. Member 1 (Mason Medina) completed the first 4 blocks of testing individually and then Member 2 (Rory McLean) provided a set of tests to further implement the test driven development cycle. This resulted in a split of labour of 120 Hours for Member 1 and 10 for Member 2.

Design Pattern

The Design pattern implemented in Catscript is Memoization which is located within the CatscriptType class in the project, This was chosen to help reduce the redunacy of opperation with Catscript. Memoization stores the return values of expensive methods within catscript within a hash map to be referenced later to prevent re-execution of already run methods. The table can then be given an argument as a key which then returns the methods return value and the key as a (key,value) pair. This improves runtime speed and lowers overall memory cost by removing duplicate objects of the same ListType.

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

Technical Writing

Catscript Guide

Introduction

CatScript is a statically typed scripting language that compiles JVM Bytecode.

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

Features

Types

Catscript is statically typed and supports basic types for variables. They can be explicitly declared and inferred.

- 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

Expressions

Catscript contains an Expression class that all expression types inherit. All expressions are code that evaluates to one of the Catscript types such as an integer, a string, or a Boolean.

Primary Expression

The default expression is a Primary expression which resolves as any expression type - Identifier

- Integer Literal

- String Literal

- Boolean Literal

- List Literal

- Null Literal

- Function Call

- Parenthesized expression

Unary Expression

Unary Expression is the Catscript implementation of the not and negative operators. It uses the (-) operator on integers and (not) for Boolean values. This allows us to negate the value in an expression.

var x = 1
print(-x) // This will print -1
not true // This returns false

Additive Expression

Additive expression is the Catscript implementation of addition and subtraction. It uses the (+) and (-) operators for addition and subtraction.

10 + 3 // This returns 13
16 - 4 // This returns 12

Factor Expression

Factor expression is the Catscript implementation of multiplication and division. It uses the (*) and (/) operators for multiplication and division.

10 * 3 //This returns 30
16 / -4 //This returns -4

Comparison Expression

Comparison Expression is the Catscript implementation of less than($<$), greater than($>$) as well as less than or equal to($<=$), and greater than or equal to($>=$).

2 $<$ 3 // true
2 $<=$ 3 // true
2 $>$ 3 // false
2 $>=$ 3 //false

Equality Expression

Equality expression is the Catscript implementation of equals and not equals. It uses the (==) and (!=) operators for assessing equality.

2 == 1 // This returns false
2 != 1 // This returns true

Statements

Catscript contains a Statement class that all statements inherit from. Unlike Expressions, statements change the state of the program instead of evaluating down to a type.

Print Statement

Print statement is used by calling the (print) keyword followed by an expression. After the interior expression has been evaluated it outputs to the Catscript output stream.

print(8) //Prints 8
var x = 'Hello Catscript!'
print(var) //Prints Hello Catscript!

Variable Statement

Variable statement is used to declare and assign variables using the (var) keyword. You can also define the type of variable using a : (colon) after the name and including the type.

var name = 'Brent'
var x: int = 7

Assignment Statement

Assignment statement allows Catscript to change the value stored within an existing variable.

var x = 10
x = 2+2
x = 'sandwich'

If Statement

If Statements in Catscript are used to determine whether or not to execute statements contained within the if. This is similar to if statements in other languages.

var x = 2
if (x == 7) {
print(7)
}
else {
print("Not 7")
}

For Statement

For Statement is similar to the for statements used in other languages employing the key word (in) to build the loop. Unlike other languages we cannot count up to a value in our for loop but iterate through lists.

```
// lists List
var list = ['a', 'b', 'c', 'd']
// iteration
for( i in list ) {
print(i)
}
```

Function Definition Statement

Function Definition Statement defines functions to be called in other parts of the code. Just like other programming languages Functions can have an explicit return type but the default is void. Functions can also have input parameters that can have explicit typing with the default being an object in Catscript.

```
function addition (x : int, y : int) : int {
return x + y
}
```

Function Call

Function Call allows for a programmer to call a function once it has been defined, if the function call matches the number and type of arguments given by the function definition it will execute. The program also verifies the symbol table to ensure the function has been registered.

```
addition(6,2)
```

Return Statement

Return Statement is used to exit a function and is necessary if an explicit return type was defined for the function. They can only be used within a function and return an expression of the correct type if the function has an explicit return type.

```
function addition (x : int, y : int) : int {
return x + y
}
```

Comments

Comments in Catscript are done using the (//) operator, putting a // in front of any line will comment it out of the code and the compiler will ignore it.

// Comments are Pretty Handy When coding

UML

The following UML diagram shows the hierarchy of the components within the Catscript system. The elements within the UML diagram are divided into Statements and Expressions which inherit from Parse Element.
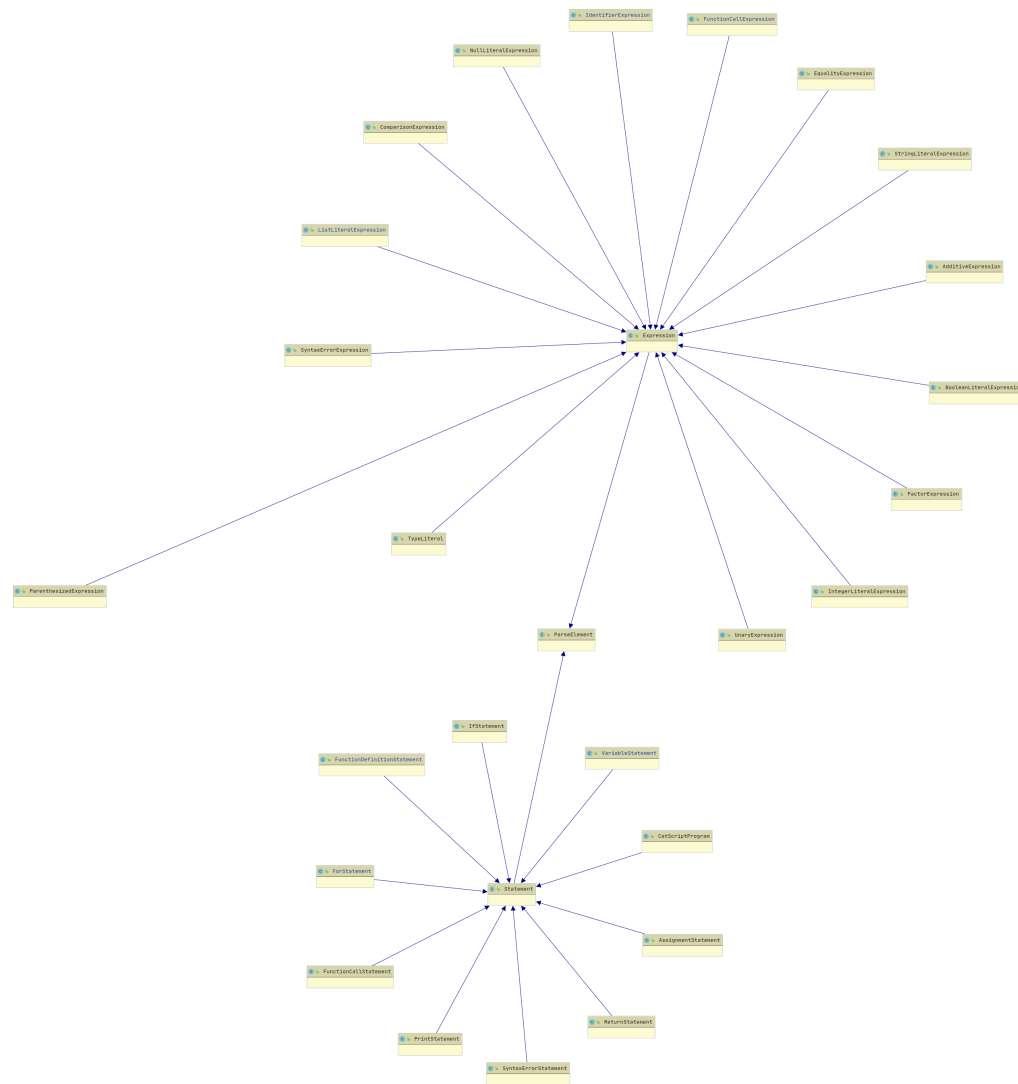
Design trade-offs

The design trade off within Catscript was the choice of recursive descent within the parser instead of using the more classically taught parser generator. This choice as made as Recursive descent is more widely used within large industry standard compilers like Java. Recursive descent mimics the recursive nature of grammar allowing for much easier modification than a parser generator and from an outside perspective allows the compiler to be understood better than the typical parser generator. While Recursive descent does have longer more written out routines its advantages make it a much better choice than a parser generator for ease of simplicity and functionality.

Software development life cycle model

The Software Development Life cycle was a Test Driven Development(TDD) model that had several checkpoints to determine completion. Each checkpoint was made up of several tests that needed to pass to ensure that the Catscript language could function as a whole. The Checkpoints were as follows; Tokenization, Parsing, Evaluation, and Bytecode Generation, Each of these checkpoints required the full completion of the previous checkpoint in order to function. All of these checkpoints were setup and verified using Git version control which allowed for ensured completion of the Test Driven Development model.