

CSCI 468: Compilers

Spring 2023

Camille Custer

Avery Jacobson

CAPSTONE Documentation

Section 1:

See attached zip "source" file

Section 2:

My team member and I shared the workload of our capstone project at approximately and 80-20 ratio. I developed the code for the entirety of the compiler, and my team member wrote challenging tests that tested the farthest edge-cases and the Catscript documentation of my progra. By working together to test our code, I can ensure that the CatScript language functions as it should. The tests are as follows:

```
no usages
@Test
void factorialRecursiveFunctionWorksProperly() {
    assertEquals( expected: "362880\n120\n", compile(
        src: "function factorial(x : int) : int {\n" +
            "    if(x == 0) {\n" +
            "        return 1\n" +
            "    }\n" +
            "    else {\n" +
            "        return x*factorial(x-1)\n" +
            "    }\n" +
            "}\n" +
            "print(factorial(9))\n" +
            "print(factorial(5))"
    ));
}
```

```
no usages
@Test
void listIterationandConcatenationWorksProperly() {
    assertEquals( expected: "420 Mary 69\n420 Had 69\n420 69\n420 Lamb 69\n420 Chops 69\n", compile(
        src: "var facts: list<string> = [\"Mary\", \"Had\", \"\", \"Lamb\", \"Chops\"]\n" +
            "for (word in facts) {\n" +
            "    print(420+\" \" +word + \" \" + 69)\n" +
            "}\n"
    ));
}
```

```

@Test
void basicLogicExpressionWorksProperly() {
    assertEquals( expected: "true\nfalse\ncan't divide by zero!\n0\nnice\nnice\n", compile(
        src: "function doubleDivide(x: int, y: int): int {\n" +
            "    if (y == 0) {\n" +
            "        print(\"can't divide by zero!\")\n" +
            "        return 0\n" +
            "    } else {\n" +
            "        return x/y/y\n" +
            "    }\n" +
            "}\n" +
            "print(doubleDivide(16,2) == 4)\n" +
            "print(doubleDivide(56,4) > 7)\n" +
            "print(doubleDivide(16,0))\n" +
            "var num1 = 64\n" +
            "var num2 = 56\n" +
            "var num3 = 2\n" +
            "if (doubleDivide(num1,num3) == 16) {\n" +
            "    print(\"nice\")\n" +
            "}\n" +
            "if (doubleDivide(num2,num3) == 14) {\n" +
            "    print(\"nice\")\n" +
            "}\n"
    ));
}

```

Section 3:

We used the memoization design pattern in our Catscript compiler. Memoization is a programming language design that creates a “cache” data structure to store the results of function calls. If a function is called again and the results have already been cached, the cached result is returned instead of recomputing the same result. An instance of the memoization design pattern in our code can be found in CatscriptType.java in the getListType() Method. This checks if the list is already stored in the “LIST_TYPES” HashMap. If it is not, it will then store it in the map. If it is, it will reference the list object in the map.

Here is an example of the code:

```

2 usages
38 private static final Map<CatscriptType, CatscriptType> LIST_TYPES = new HashMap<>();
9 usages Carson Gross
39 @ public static CatscriptType getListType(CatscriptType type) {
40     CatscriptType listType = LIST_TYPES.get(type);
41     if (listType == null) {
42         listType = new ListType(type);
43         LIST_TYPES.put(type, listType);
44     }
45     return listType;
46 }

```

Section 4:

Catscript Introduction

Catscript is a statically typed functional programming language. It is a simple scripting language, here is an example:

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

Catscript Type System

The Catscript type system consists of 6 different types:

int- a 32 bit integer

- Catscript example:

```
int position = 0;
```

string - a java-style string enclosed by quotes

- Catscript example:

```
String name = "Carson";
```

bool - a boolean value either true or false

- Catscript example:

```
bool isAtEnd = false;
```

null - the null type

- Catscript example:

```
int position = 0;
```

object - any type of value

- Catscript example:

```
Object arg = args[i];
```

List Literal

- List literals begin with a left bracket "[" and are followed by an expression. The expression is followed by 0 or more expressions of the same type. A list can contain lists.
- Catscript example:


```
["cat"]
[9, 8, 7, 6]
List<Object> args = new ArrayList<>();
```
- In Catscript, lists can be iterated through as follows:


```
// lists List<Integer>
var lst = [1, 2, 3];

// iteration
for( i in lst ) {
    print(i);
}
```

Statements

For loops

- In Catscript, for loops are identified with the starting word "for". Any code inside the following parentheses is the condition for the loop. The code inside the curly braces is what is to be iterated over.
- Catscript example:


```
for(Statement arg : arguments){
    //code to be executed
}
```

If Statements

- If statements are identified with the starting word "if". Any code inside the parentheses is the condition to be checked. If the condition is true, the code inside the curly braces will be executed. If it is not true, either the statement will break or the "else" expression will be executed if it exists.
- Catscript example:


```
if( x > 10 ) {
    print(x)
}
```

Print Statement

- Print statements are identified with the word "print". Anything inside the parentheses is outputted.

- Catscript example:

```
print("This is an example");
```

Var Statement

- Var statements are used to assign values to variables. The value following the keyword "var" is used as the identifier for the var statement. The right-hand side of the equal sign is the assigned value to the identifier.
- Catscript example:

```
var x = 100;
```

Function Call Statement

- Function call statements are used to call functions using the identifier for that function. Function calls can take parameters but they are not required.
- Catscript example:

```
foo()  
foo(x)  
foo(x, y, z)
```

Return Statement

- Identified with the word "return". If there is nothing following the word, the function will break and will go to where it was called. If there is something following the return, then that value is returned to the function where it was called.
- Catscript example:

```
return true;  
return;  
return x;
```

Expressions

Equality Expression

- Equality expressions consist of a comparison expression followed by either "==" or "!=" and then another comparison expression. This will return either "true" or "false"
- Catscript example:

```
1 == 1 //true  
1 != 1 //false
```

Comparison Expression

- Comparison expressions in Catscript are used to compare the left-hand and right-hand sides of the expression. Using operators "<", ">" ">=", "<=", this expression will either return "true" or "false".
- Catscript example:

```
1 > 2 //false
1 <= 10 // true
```

Additive Expression

- Identified by the operators "+" and "-", additive expressions are used to add or subtract two values together or concatenate two string values. String values can not be concatenated with the "-" operator.
- Catscript example:

```
a = 5 + 5 //a = 10
x = 10 - 5 //x = 5
String name = "Cat" + "script" //name = Catscript
```

Factor Expression

- Identified by operators "*" and "/", factor expressions are used to multiply or divide the left-hand and right-hand sides of the expression.
- Catscript example:

```
int x = 4 * 3 //x = 12
int y = x / 2 //y = 6
```

Unary Expression

- Unary expressions are identified with a minus sign "-" and the word "not". They are used to negate values of type boolean or integer.
- Catscript example:

```
not true //false
-(-8) //positive 8
```

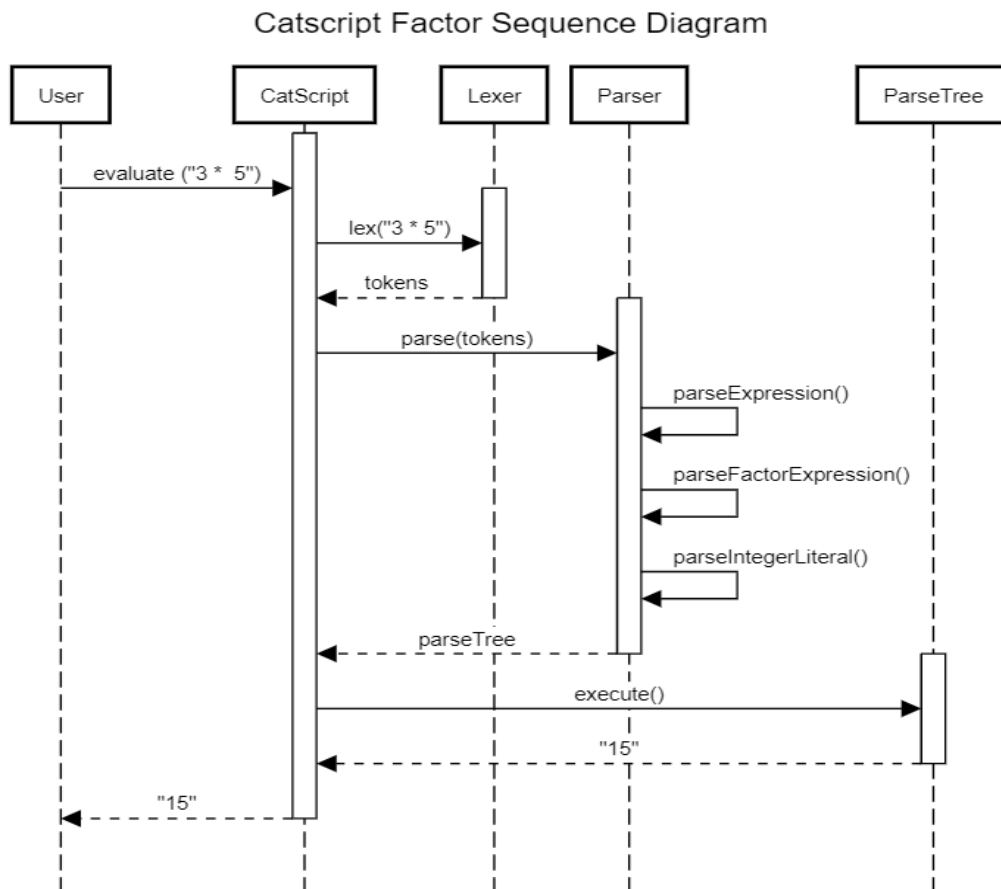
Primary Expression

- A primary expression can be any one of the following:
- Identifier, string, integer, boolean, null, list literal, function call, or parenthesized expression
- Catscript example:

```
( 2 + 2 )
false
null
7
```

Section 5:

The following sequence Diagram shows the process of the CatScript Parser when compiling `(3*5)`. You can see how the Parser tokenizes, lexes, parses, and compiles the statement.



Section 6:

The Catscript language relies on Recursive Descent Parsing. Recursive Descent parsers are fairly easy to implement and debug. This method of parsing works well for Catscript, but often run slowly compared to other parser generators when paired with large or complex grammars. We could of implemented a parser generator instead which would greatly improve parsing performance, but in turn are rather complex to implement and leave the developer with less control over the parsing process.

Section 7:

The software development lifecycle model that we used is the Test Driven Development model. This method is a Linear sequential model that is broken down into tests that make up stages. Each test within the stage is completed before moving onto the next stage. In our Catscript Parser these stages were as follows: Tokenization, Expression Parsing, Statement Parsing, Evaluation testing, and Compilation.

These stages were verified using testing functions that ensured our algorithm worked properly. This model helped my team as we were unable to move forward without completely polishing the aspects of the compiler that would become dependent as we advanced onwards. This hindered our progress as well, since we may of not been able to directly take what we learned from lecture and put it into our code, as we were still working on tests that were weeks behind what was being covered in lecture.