

Andrew Cilker  
Carson Gross  
CSCI 468  
6 May 2022

## Compilers Capstone Portfolio

### Section 1 – program files

Link to the program: (as zip file)

### Section 2 - Teamwork

The work done in this project was separated into four different checkpoints: tokenizer, parsing, evaluation, and bytecode. Code creation was up to each individual member, but partners were responsible with creating an additional testing suite to display performance criteria of the compiler. This project had a large focus on test-driven development and all the debugging that comes along with it. Both team members came up with unique tests to aide in assessing the completeness of the other person's compiler.

Total Hours to completion: ~100 hours

Member 1:

Software developer

~80 hours

Member 2:

Software tester and documentation

~20 hours

Teamwork: The my team worked on this capstone project was separated by development and testing. I focused on the development side of this project and my team member focused on the testing. My team member 2 helped me by writing three software tests to aide in developing my own compiler. The development took roughly 95% of the time spent on the project, and the software testing took about 5%.

### Section 3 - Design Pattern

The design patter used in this capstone project is the Memoization Pattern. This design pattern is typically used to increase program efficiency by limiting the amount of new objects being created. Implementing this design allows the function to only invoke new CatScriptType objects when there is not already an object of that type instantiated. For longer streams of code, this is necessary to keep memory usage down by limiting needless redundancy.

This code can be found in CatscriptType.java

```
static final 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 Documentation

### Introduction

Catscript is a statically typed programming language that compiles into bytecode. The goal of Catscript is to be a fun programming language with a cool name that pulls elements from different languages. It also uses a recursive descent parser

### Features

#### Type Literals

Catscript supports basic types also found in Java.

- Int
- string
- bool
- list
- null
- object

#### if statements

if statements in Catscript are basically the same as if statements in java. They recognize “if”, “else if”, and “else” and if the arguments in parenthesis evaluates to true, the code runs.

```
If(1 == 1){  
  print(“true”)  
}
```

#### For loops

For loops in Catscript are very similar to for loops in Python

```
list[1,2,3,4,5]  
for (I in list){  
  print(“i”)  
}
```

```
//Prints 1 2 3 4 5
```

#### Function Definitions

Put “function” in front of an identifier to invoke a function definition.

```
Function foo(str: String){  
  print str
```

```
}
```

## Unary Expressions

Catscript has “-” to make negative values and a “not” keyword to flip boolean values

## Comparison

Catscript has all the following comparison functionality: less than, greater than, less than or equal, greater than or equal.

```
0 < 1   → true
0 > 1   → false
0 <= 1  → true
0 >= 1  → false
```

## Equality

Catscript can check for equality or not equal for a boolean output with “equal equal” and “bang equal”

```
0 == 1 → false
0 != 1 → true
```

## Variable Statements

Catscript can define an object’s type with the use of a “:” between the variable and the <type>

```
var flag : bool = false
var name : String = “Sally”
var num : int = 1
```

This is not required because catscript will also handle implicit typing

## Section 5

Include and discuss one of the UML diagrams included in this directory:  
(Parse Elements)

## Section 6

The largest design decision made for the Catscript project was to use a Recursive Descent parser.

Discuss recursive descent vs a parser generator

## Section 7

Describe the model that you used to develop your capstone project. How did this model help and/or hinder your team?

We are using Test Driven Development (TDD) for this project

- Section 4: Technical Writing - Include the documentation generated by your partner for the catscript programming language**
- Section 5: UML - Include and discuss one of the UML diagrams included in this directory**
- Section 6: Design trade-offs - You must write this, maybe discuss recursive descent vs. a parser generator?**
- Section 7: Software development life cycle model - We did Test Driven Development, please discuss your experience with it**

Section 4: Technical writing. Include the technical document that accompanied your capstone project.

Section 5: UML.

Include a UML diagram for parse elements

Section 6: Design trade-offs

To be discussed later in the class

Section 7: Software development life cycle model

Describe the model that you used to develop your capstone project. How did this model help and/or hinder your team?

We are using Test Driven Development (TDD) for this project