

CSCI 468: Compilers Capstone Portfolio

Hannah Madsen
Montana State University
May 2022

Section 1: Program

Source Code:

<https://github.com/hbmadsen/csci-468-spring2022-private/blob/master/capstone/portfolio/source.zip>

Section 2: Teamwork

Kieran and I contributed equally to this project. We both individually wrote a recursive descent parser that converted Catscript to bytecode. We also provided each other with three tests to include in our test driven development. Lastly, we each wrote up documentation of Catscript and how it worked, this documentation was exchanged between us to ensure that we both had a sound understanding of the language.

Kieran's tests:

<https://github.com/hbmadsen/csci-468-spring2022-private/blob/master/src/test/java/edu/montana/csci/csci468/partnerTests.java>

Section 3: Design Pattern

Memoization was employed in the CatscriptType class for the getListType() function, where type for the given list is returned. Memoization is implemented through the use of a hashmap to cache previous list types, so new list types do not need to be created every time the function is called.

```
private static Map<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 below documentation was provided to me by my partner, Kieran Ringel. Also see Ringel_Catscript.md here:

https://github.com/hbmadsen/csci-468-spring2022-private/blob/master/capstone/Ringel_Catscript.md

Catscript Guide

Introduction

Catscript is a scripting language similar to python. It is an interpreted language so the source code is converted to bytecode which is then run by the java virtual machine.

Features

For loops

A for loop is used to iterate over an expression. The for loop in catscript operates similarly to the for loop in python.

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

If statement

If statements support conditional logic. If statements are written using the keyword "if". Catscript also supports else if and else statements.

```
if (1 > 2) {  
    print("wrong")  
} else {  
    print("right")  
}
```

Print statements

Print statements are used to print strings to the output.

```
print("Hello world")
```

Variable Statements

Variable statements are used to declare variables and can optionally include the variables type.

```
var x = 1  
var y : int = 2
```

Assignment Statement

Assignment statements are used to assign a new value to a variable.

```
x = 2
```

Function Declaration

A function is defined using the "function" keyword.

```
function foo() {  
    print("bar")  
}
```

Parameters can be passed into a function, they are specified in the parentheses. Multiple parameters must be separated by commas.

```
function foo(x, y : int) {  
    print(x)  
    print(y)  
}
```

The function declaration can also include the type, specifying what the return type is.

```
function foo(x : int) : int {  
    return x  
}
```

A function's body can consist of statements and return statements.

Equality Expressions

Equality expressions are used to determine if two values are equivalent.

1 == 1 would return true, or not equivalent 1 != 2 would return true.

Equality expressions return a boolean.

Comparison Expressions

Comparison expressions can compare two integer values and determine if the left value is greater than, greater or equal to, less than, or less than or equal to the right value.

```
1 < 2  
1 <= 4  
5 > 3  
5 >= 4
```

A comparison expression returns a boolean.

Additive and Factor Expressions

Additive and factor expressions are used to perform basic mathematical operations.

```
1 + 2
```

3 - 2
4 * 5
4 / 2

Unary Expression

A unary expression is used to negate a numerical value or boolean.
not true would result in false - 1 results in -1.

List Literal

Lists are used to store multiple variables in one item.

[1, 2, 3]

Types

Catscript supports integers, strings, booleans, objects and lists.

Section 5: UML

See class diagram on page 6.

Section 6: Design Trade-offs

The design decisions and corresponding tradeoffs made for the compilers project included choosing to use a recursive descent parser and to not follow a visitor pattern for evaluation and compilation. We chose to use a recursive descent parser rather than a parse generator because recursive descent is simpler and provides a better understanding of the recursive nature of grammars. The tradeoffs for choosing recursive descent were that it required more code and that it is not as standard as parser generators. Additionally, we made the decision to not use the visitor pattern that was used in the *Crafting Interpreters* book to address the expression problem. In our project, we had evaluation and compilation take place within each parse tree node, which was directly embedded in the parse tree and made it easy to implement and kept everything in one place. The tradeoff with this approach was that we did not separate concerns, which is often preferred.

Section 7: Software Development Life Cycle Model

Test Driven Development (TDD) for this project. This model was very helpful because it ensured that the prerequisites were fulfilled for each part of the compiler, thus, avoiding errors in later stages of development. However, I was quite reliant on these tests and found that tests would pass in some stages of development, but failed to ensure I had properly implemented everything with the whole picture in mind. So, I had to revisit and rewrite code that I had previously thought was correct.

