

Montana State University
CSCI - 486 Compilers Capstone Portfolio
Spring 2022

John Jubenville
Ryan Kraus

Section 1 : Program

See the attached *sourcecode.zip* for source code. The source code can be compiled using the IntelliJ IDE.

Section 2 : Teamwork

Team Member 1:

Team Member 1 was the primary developer for the project. Team Member 1 used the test scripts provided by the professor and Team Member 2 to write the Java code necessary to create a functional compiler for the Catscript language. Team member 2 started with developing the parser, then moved on to developing the statements and expressions for the language, and finally the bytecode compiler.

Estimated Time: 190 Hours

Estimated Percentage: 95%

Team Member 2:

Team Member 2 was the primary test and documentation writer for the project. Team Member 2 wrote a small suite of tests to test the compiler written by Team Member 1. Team Member 2 also provided the technical documentation for the project. The technical documentation broke down the Cat Script language usage and defined the Catscript statements and expressions.

Estimated Time: 10 Hours

Estimated Percentage : 5%

Section 3 : Design Pattern

One design pattern that was used was the Memoization Pattern. The purpose of the Memoization Pattern (also called the flywheel pattern) is to limit the amount of new objects being created or processes being run by caching objects that will be called repeatedly. By caching frequently called objects, a program saves the processing time that goes into creating the objects. Memoization also saves memory by only creating a single instance of an object, instead of one for every time that it is called.

In this case the Memoization Pattern was applied to the “getListType” function in the CatscriptType Class. The function returns a ListType object that corresponds to the parameter CatscriptType. Prior to applying the Memoization Pattern, the function would initialize a ListType every time it was called. This function is called frequently so the initialization of the ListType could add substantial time and memory to a program run.

To apply the Memoization Pattern, a static HashMap was created in the class that holds a CatscriptType and the corresponding ListType. When a new type is run through the getListType function, a ListType is initialized and then stored in the HashMap. In any subsequent calls to the getListType function, the value of the ListType is pulled from the HashMap, rather than reinitialized in the function. This saves the program the time and memory required to initialize the ListType.

A copy of the getListType function is included here for reference:

```
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;
}
```

Catscript Guide

Introduction

Catscript is a statically typed scripting language. Here is an example:

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

Output : Hello World

Features

Identifier Expressions

Objects of any type can be assigned to identifier variables:

```
var x = 1  
print(x)
```

Integer Literal Expressions

Integer Assignment:

```
int x = 1
```

String Literal Expressions

String Assignment:

```
string x = "Hello"
```

Boolean Literal Expressions

Boolean Assignment:

```
bool x = true
bool y = false
```

List Literal Expressions

Lists contain objects of any type in Catscript:

```
list x = [1, 2, 3]
list y = ["hi", "there"]
list z = [true, false]
```

Null Literal Expressions

Catscript allows for null assignment:

```
var x = null
```

Additive Expressions

Catscript allows for integers to be added or subtracted. Strings can be concatenated:

```
var x = 1 + 1
var y = 1 - 1
string s = "Hello" + " World"
```

Factor Expressions

Catscript allows integers to be multiplied or divided:

```
var x = 1 * 1
var y = 1 / 1
```

Comparison Expressions

Comparing integers in Catscript:

```
bool x = 1 > 2
bool y = 1 >= 1
bool z = 1 < 2
bool a = 1 <= 1
```

Equality Expressions

Catscript allows for equality testing with any type:

```
bool x = (1 == 1)
bool y = ("Hi" == "Hi")
```

Unary Expressions

Integers can be negative or negated

```
int x = -1
bool y = not true
```

Parenthesized Expressions

Catscript allows for parenthesized expression:

```
int x = (-1 + 3) / 2
```

Function Call Expressions

Define functions and optionally include parameters:

```
function add(x, y) {
    return x + y
}
var x = add(1, 1)
```

Assignment Statement

Variables can be assigned

```
var x = 1
```

For Statement

Catscript allows for the use of the logical For loop:

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

Functional Call, Function Definition and Return Statements

Define a function with optional parameters and define a return value:

```
function add(x, y) {
    return x + y
}
var z = add(1, 1)
```

```
function subtract(x : int, y : int) : int {  
    return x - y  
}  
int z = subtract(1, 1)
```

If Statement

Catscript allows for the use of the logical If/Else loop:

```
int x = 5  
if(x > 0){  
    print("x is positive")  
} else {  
    print("x is negative")  
}
```

Print Statement

Print statement to console

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

Output:

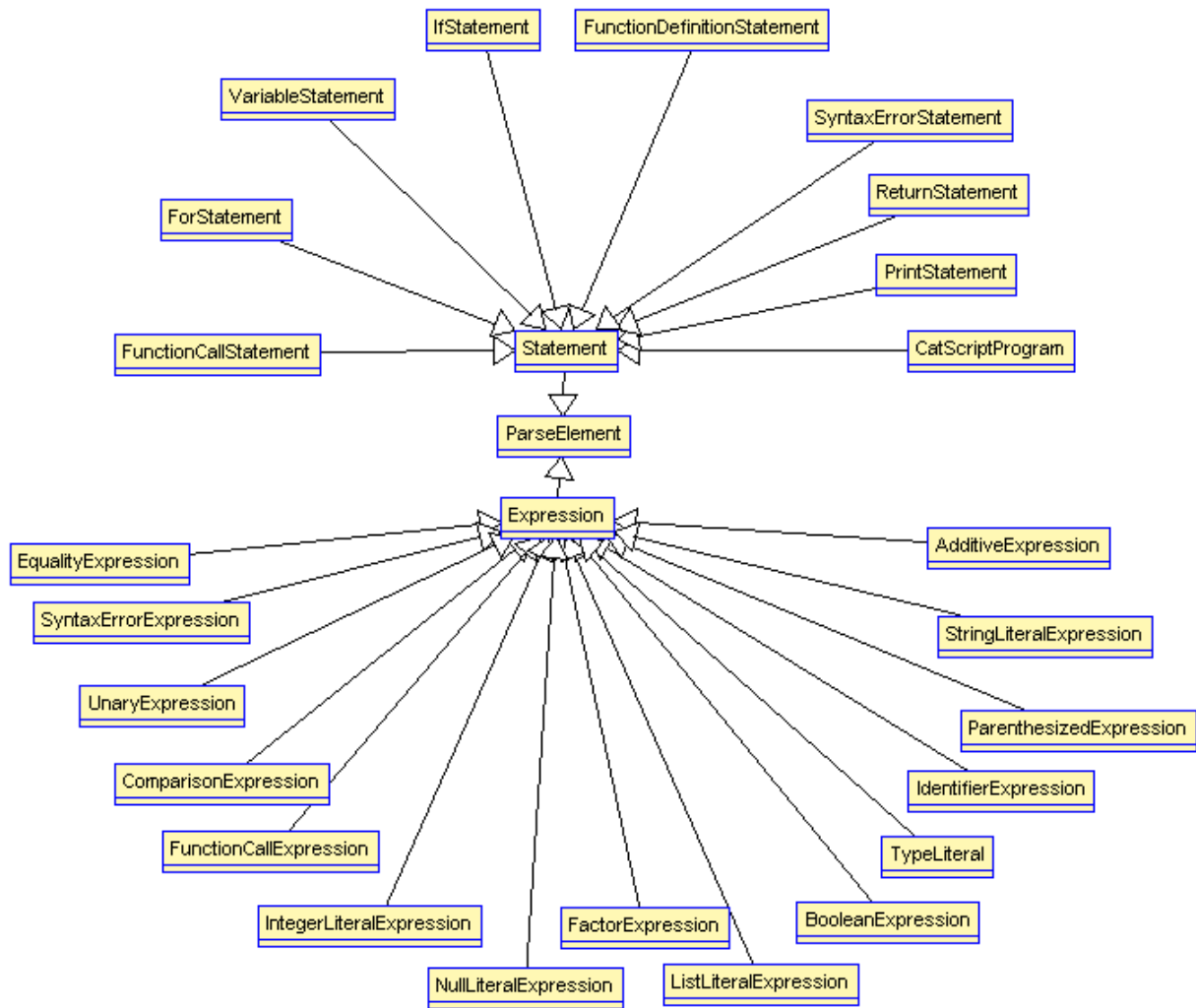
```
1  
2  
3
```

Variable Statement

Variable statements assign type based on implied type:

```
var x = 1
```


Section 5 : UML



Section 6 : Design Trade-offs

The major design tradeoff in this project was the choice to use Recursive Descent over Parser Generation for the compiler. This choice was ultimately made by the professor, but it had large implications on the development of the Catscript compiler.

Parser Generators are complicated to write, and the code can be highly obfuscated. They also become slower as the amount and variety of what can be parsed increases. It is difficult to implement error handling and multi-line commenting using a parser generator, which can make things difficult for the end user as well. Parser Generators can make small projects quicker, but use of a parser generator is likely not allowed in a compilers class as it removes too much from the education of how parsers and compilers work.

Recursive descent is more widely used in industry, and has a more intuitive design and process flow. It is the ideal method for education as it teaches students about the fundamentals of language design and provides a clearer picture of what parsing and compiling are. For these reasons, Recursive Descent was chosen as the design for the compiler.

Section 7 : Software Development Life Cycle Model

The primary development model used in this project was the Test Driven Development model. The Test Driven Development model works by converting software requirements to test cases, and then writing the code necessary to pass the tests. In this case the tests were developed by the professor and Team Member 2, then Team Member 1 worked to complete all of the tests.

One advantage of the Test Driven Development model is that it breaks large projects, such as this one, into much smaller and manageable chunks. The developer is forced to focus on solving one test at a time, making large projects more doable. It also can lead to more efficient code, since they are only writing enough code to pass the tests. Another advantage is that it leads to finer debugging, since each test can test a single component in code, rather than the whole project put together.

Disadvantages of Test Driven Development include the need to write full coverage tests prior to developing the program. In cases where the software requirements are vague or unknown, this can be almost impossible. In the case of the compiler however, the input and output of each compiler component is well defined and fairly easy to get full coverage on. Also this step was taken care of by the professor.

From an education standpoint, following the Test Driven Development model was a great way to learn the ins and outs of a compiler without getting overwhelmed by the complexity of such a large project.