

Montana State University Computer Science Department

Senior Team Portfolio

CSCI 468 Compilers

Spring 2023

Alexander Fischer, Devan W. Eastman-Pittam

Section 1 Program:

You can find the project files in a zip file contained in the /capstone/portfolio directory of my csci-468-spring2023-private github repo.

Section 2 Team Work:

For this project we had four major sections that we worked on during the semester. The first was tokenization and being able to break apart code into readable tokens. The second was working on parsing to be able to understand the tokens we produced during tokenization. The third was evaluating the parsed tokens and statements. And finally was generating bytecode for outputting results. Each group member wrote tests for the other, which can be found in the byte code test area. Both group members spent roughly 20 hours working on tests for the other member and assisted in their ability to complete them. Team members 1 and 2 had roughly equal contribution in the group work going towards completion of the project. Tests are in the bytecode test folder.

Section 3 Design Pattern:

```
static final HashMap<CatscriptType,ListType> hmap = new HashMap<>();

public static CatscriptType getListType(CatscriptType type) {
    ListType maptype = hmap.get(type);
    if(maptype==null){
        ListType lstype = new ListType(type);
        hmap.put(type,lstype);
        return lstype;
    }else {
        return maptype;
    }
}
```

For our project we used the memoization design pattern in our code as it fit the needs of our project the most.

Memoization is the process of adding repeated data requests to a certain cache location, and accessing the result from the cache in order to get the result instead of constantly calculating the same result of a function over and over again. This limits the total object creation amount and speeds up the program.

Section 4 Technical Documentation / Partners Catscript

Documentation:

Introduction

Catscript is a statically typed scripting language that is compiled to Java Virtual Machine (JVM) bytecode. It is designed to be simple, and efficient.

Comments

Comments are used to document your code and make it more readable. In Catscript, you can add comments using the two forward slash characters `//` followed by your comment.

For example:

```
//This is a comment in Catscript
```

Variables

To declare a variable in Catscript, you must use the `var` keyword followed by the variable name and variable type if required.

For example:

```
var x = 1 //Infer the variable type  
var y : int = 2 //Explicitly Define the variable type
```

Catscript supports the following variable types:

- int for integers values
- string for string values
- bool for Boolean values
- list for initiating a list of values
- null for null values
- object to create an instance of an object

Print Statements

Catscript has a built-in statement to print directly to the console. You can use `print()` to print the value(s) you would like.

For example:

```
print("Hello World!")
```

Math Operations

Catscript supports basic math operations such as: addition '+', subtraction '-', multiplication '*', and division '/'

For example:

```
print(10 + 1) //output = 11  
print(10 - 11) //output = -1  
print(10 * 2) //output = 20  
print(10 / 2) //output = 5
```

Comparison Operations

Catscript supports greater than '>', less than '<', greater than or equal to '>=', and less than or equal to '<=' operators.

For example:

```
print(4 > 5) //Output: false
print(4 < 5) //Output: true
print(4 >= 5) //Output: false
print(4 <= 5) //Output: true
```

Equality Operations

Catscript supports '==' and '!=' operations for comparing equality and not equality

For example:

```
print(4 == 5) //Output: false
print(4 != 5) //Output: true
```

Unary Expressions

Catscript has two ways to negate values depending on type: '-' for variables and 'not' for Boolean values.

For example:

```
var x = 5
var y = -10
print(x + y) //Output: -5
var z = true
print(not z) //Output: false
```

For-Loops

Catscript supports for-loop statements using the 'for' keyword. You can iterate over each item in a list using the 'in' keyword

For example:

```
var list = [1,2,3]
for (x in list){
    print(X)
}
```

If-Else Statements

Catscript supports if-else statements for conditional branching. You can use the 'if' keyword followed by an expression and the body of code to run within your if statement. You can also use 'else if' and 'else' to specify alternate conditions if the first condition is not met.

For example:

```
var x = 10
if (x < 5){
    print("x is less than 5")
}
else if (x < 10){
    print("x is less than 10, and greater than or equal to 5")
}
else{
    print("x is greater than or equal to 10")
}
```

While Loops

The while loop is used to execute a block of code if a certain condition is true. The condition is checked before each iteration of the loop.

For example:

```
var x = 1
while (x < 10){
    print(x)
    x = x + 1
}
```

In the above code you can see that x will increase by 1 for each iteration of the loop until x is equal to 10.

Functions

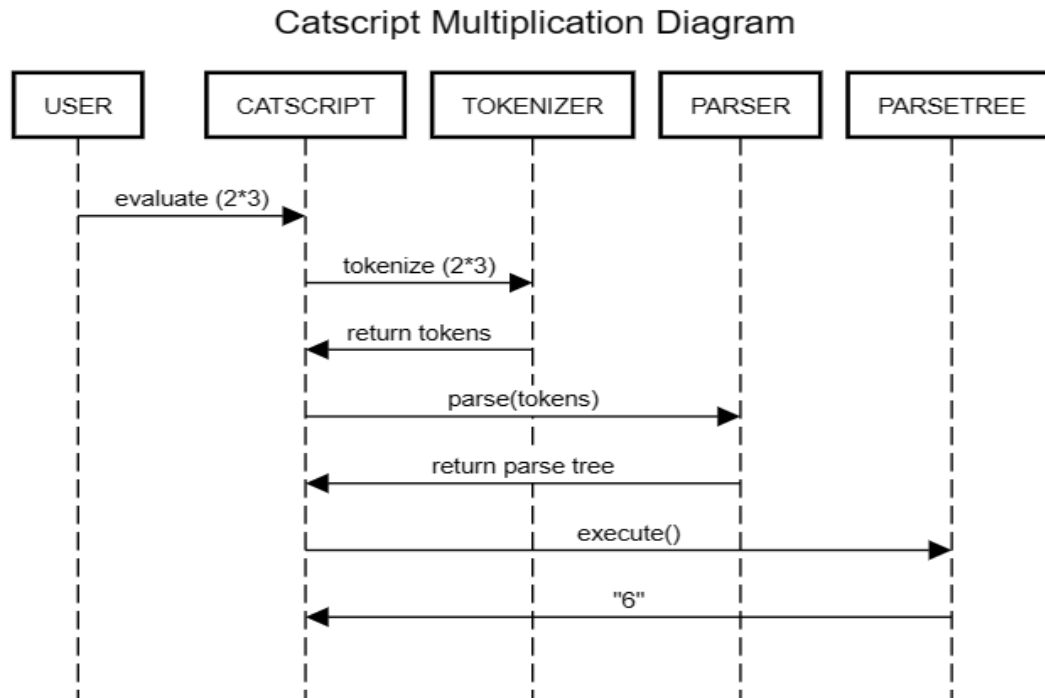
Functions are used to encapsulate a block of code and make it reusable. In Catscript, functions are defined using the function keyword followed by the function name and a set of parentheses containing any parameters that the function accepts.

For example:

```
function add(x,y){
    return x + y
}
var result = add(3,5) // Function call with passed values
print(result) //Output: 8
```

As seen above, the add function accepts two parameters x and y, adds them together, and returns the result.

Section 5 UML:



Here is an example of how the catscript compiler functions for multiplication, in it it shows the steps of sending requests for multiplication, tokenizing, parsing, executing, and returning results.

Section 6 Design Trade Offs:

For this project we used recursive descent parsing which is a top down style parser instead of bottom up parsing which had quite the large effect on the design decisions made while working on project development.

Recursive descent is useful as you can better add to the process during development working with tokens directly as your needs evolve instead of working with expressions and grammars, but can cause issues as it is quite in depth to work with.

Section 7 Development Model:

For this project we used the test driven development model TDD. Carson developed different tests for each portion of the project along the way and the end goal was to have a complete compiler by the end of the semester.

Some benefits were that we had a clear goal in mind while working on the project and that we could gradually progress along the way without having sudden changes to expectations or assignments.

Some issues going with this style of development were that we were shoehorned into certain coding decisions that we would not choose to fit the existing tests even if another method could present comparable results.