

# **CSCI 468 - Compilers**

## **Spring 2022**

**Max Lineberger**

**Tanner Rubino**

# Section 1: Program

---

Zip file of source code is included in the source.zip file in the /capstone/portfolio directory:

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

## Section 2: Teamwork

---

Our team consisted of a primary coder, who completed the majority of the code and worked to get all of the tests provided by the professor to pass as well as a test writer, who wrote a series of tests to check different edge cases that could occur in the program worked to provide complete documentation of the Catscript language.

- Tanner Rubino- Primary coder, wrote the majority of source code and completed sections on design tradeoffs, software development life cycle, design patterns, and UML diagrams.
- Max Lineberger- Primary tester, wrote a series of tests for the primary coder to check unique situations in code and completed the Catscript Documentation section of this portfolio.

## Section 3: Design pattern

---

The memoization design pattern was used in the CatscriptType.java file located at the path

csci-468-spring2022-private/src/main/java/edu/montana/csci/csci468/parser/CatscriptType.java Starting at line 36:

```
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 new ListType(type);
}
```

This pattern is used so that every time the function getListType() is called it can rely on a cache of already created ListTypes instead of creating a new ListType each time. This design pattern allows for a much more efficient use of memory. If a ListType was already created and used at one point in the program then it will be able to use that already created ListType again from the cache without needing to create a new type and store it in memory each time.

## Section 4: Technical Writing

---

### Introduction

---

Catscript is a simple programming language that uses common features found in many other coding languages. Here is an example of code you can write in Catscript:

```
var x = "hello "
var y = "world"
print(x + y)

output:
hello world
```

### Features

---

#### Types

There are different types of values in Catscript, and it is expressed in the grammar as:

```
type_expression = 'int' | 'string' | 'bool' | 'object' | 'list' [, '<' , type_expression, '>']
```

The different types are:

- int: Stores integer values (ex. 1, 2, 0, -38, 45366)
- string: Stores combinations of characters (ex. "hello world", "h3ll0 w0rld")
- bool: Stores a value of true or false
- object: Stores values for any other type and is generic
- list: Stores a collection of values of a single type (ex. list\, list\). Lists cannot change their type after creation.

## Expressions

There are different types of expressions in Catscript, and they are expressed in the grammar as:

```
expression = equality_expression;

equality_expression = comparison_expression { ("!=" | "==") comparison_expression };

comparison_expression = additive_expression { (">" | ">=" | "<" | "<=" ) additive_expression };

additive_expression = factor_expression { ("+" | "-" ) factor_expression };

factor_expression = unary_expression { ("/" | "*" ) unary_expression };

unary_expression = ( "not" | "-" ) unary_expression | primary_expression;

primary_expression = IDENTIFIER | STRING | INTEGER | "true" | "false" | "null" |
                    list_literal | function_call | "(" , expression , ")"
```

Expressions are evaluated during runtime, and are used for most of the logical functions of Catscript. They mostly handle basic math and comparisons.

### Equality Expression

The equality expression is used to determine if two values are equal or not equal. This could be used to check if two numbers, booleans, or strings are the same. This will return a boolean value.

Example: `1 == 1` , `true != false`

### Comparison Expression

The comparison expression is used to determine if two numbers are less or greater than another. It also has *less than or equal to*, and *greater than or equal to*. This will return a boolean value.

Example: `5 > 3` , `6 <= 2`

### Additive Expression

The additive expression is used to add two numbers, two strings, or a number and a string together. This will return a number or a string depending on the values given. If they are both numbers it will return a number, otherwise it will return a string.

Example: `1 + 1` , `5 + " , I like that number" , "hello" + " world"`

### Factor Expression

The factor expression is used to multiply and divide two numbers. It will return the value calculated.

Example: `2 * 2` , `9 / 3`

### Unary Expression

The unary expression is used to negate a boolean, and make a number negative. It will return the "opposite" of the value given.

Example: `not true` , `-1`

### Primary Expression

The primary expression is used for single variables. These will evaluate to the same value that is passed in.

Example: `1` , `hello world` , `true`

## Variables

### Defining Variables

Variable statements are expressed in the Catscript grammar as:

```
variable_statement = 'var', IDENTIFIER,
                    [ ':' , type_expression , ] '=' , expression;
```

Variables start with 'var' and then are identified by user choice. This could be a word or a character. Example: `var one : int = 1` The type can be assigned by using a colon, but this is optional. On the right side of the equal sign will be the value of what the identifier will be assigned to. If there is no type given at declaration, then it will automatically assign the same type as the expression. Example: `var x = "hello world"`

## Assigning Variables

Assignment statements are expressed in the grammar as:

```
assignment_statement = IDENTIFIER, '=', expression;
```

Once a variable is declared, it can be changed to a value of the same type. This is very similar to defining variables, but no longer needs to be declared as a 'var'.

Example:

```
var one : int = 1
one = 2
```

## Printing

Print statements are defined in the Catscript grammar as:

```
print_statement = 'print', '(', expression, ')'
```

Print statements display an expression in the output of the code. They start with 'print' and will output the expression within the parentheses.

An example of a print statement:

```
print(1 + 1)
```

Output:

```
2
```

## Functions

### Function Declaration

The function declaration statement is expressed in the grammar as:

```
function_declaration = 'function', IDENTIFIER, '(', parameter_list, ')' +
    [ ':' + type_expression ], '{', { function_body_statement }, '}';
```

Functions start with the keyword 'function' and are followed by an identifier of the user's choice. Then, in parentheses, there is a list of parameters needed to be given as input to the function. Optionally, a return type can be added to the function.

Example:

```
function add(x : int, y : int) : int{
    return (x + y)
}
```

If there is no return type given then it will be void and not directly return any values. This could be used to print statements or assign a variable.

Example:

```
function whatIsTheValue(x){
    print(x)
}
```

The body statement will be executed when the function is called.

### Function Calls

A function call is expressed in the grammar as:

```
function_call = IDENTIFIER, '(', argument_list , ')'
```

A function can be used by calling it by name, and then sending in the arguments needed in parentheses (in the form of a list).

Example:

```
num = add(1,2,3)
print(num)
```

Output:  
6

## For loops

The for statement is expressed in the grammar as:

```
for_statement = 'for', '(', IDENTIFIER, 'in', expression ')',
'{' , { statement } , '}' ;
```

It is used to iterate through a list, and then the user can do something with each value in the list, such as print it out. It starts with 'for' and then in parentheses will be an identifier, the keyword 'in', and a list marked as an expression. The identifier will be only used for this for statement, and will be assigned to each part of the list, one at a time.

For each iteration, it will execute the 'statement' which is within curly brackets after the for statement.

Example:

```
var list = ["hello", "world"]

for(word in list){
    print(word)
}
```

Output:  
hello  
world

## If Statements

If statements are expressed in the grammar as:

```
if_statement = 'if', '(', expression, ')', '{',
{ statement },
'}' [ 'else', ( if_statement | '{', { statement }, '}' ) ] ;
```

If statements are used to check conditional statements and rely on boolean values. They start with 'if' and then a boolean expression in parentheses. If the expression evaluates to true, then the statement in curly brackets will be executed.

Example:

```
var x = true

if (x == true){
    print("TRUE")
}
```

Output:  
TRUE

There can optionally be an 'else' statement, following the same format but optionally containing another if statement within it. If the 'if' statement is false, then the 'else' statement will be executed instead.

Example:

Output:  
FALSE

### UML Class Diagram for Expressions in Catscript:

We used Test Driven Development (TDD) for this project. A suite of tests were created by the professor and we had to write our code to pass every test. The TDD model provided an excellent framework to completing the project, by giving clear goals and clear functions for the code to implement. The use of TDD greatly aided in ensuring that every piece of the code worked correctly and helped in covering edge cases and unique situations that the code must handle. This model also helped ensure that goals were met within a specific time over the course of the semester, as each major part of the project had a test suite associated with it that needed to be completed by specific dates.