

## Section 1: Program

## Section 2: Teamwork

As team members, me and Justin Scarbrough wrote each others documentation and supplied each other with 3 tests to further improve our compilers. My partner gave me tests to confirm the ability of my code to perform nested for loops, string concatenation, and variable assignment.

## Section 3: Design pattern

The design pattern used in our project is memoized type access inside the `catscriptType` file. When asked for a certain list type, a hashmap will be searched to determine if there already exists a list type for the asked type. We used memoization so that we aren't executing the same data multiple times.

## Section 4: Technical Writing: Catscript Guide

This document contains information and usage for the Catscript programming language.

### Introduction

Catscript is a lightweight programming language which shares features with similar scripting languages. Catscript is a strong (statically) typed language which supports the following types:

- `int` - a 32 bit integer, such as `1`
- `string` - a java-style string, such as `"string"`
- `bool` - a boolean value, such as `true`
- `list` - a list of value with the type `'x'`, such as `[1,2,3]`
- `null` - the null type
- `object` - any type of value

### Features

#### Arithmetic

Math in Catscript follows a simple pattern:

- Additive Expressions: `+` or `-`
  - Factoring Expressions: `*` or `/`
- Assignment is denoted with `=`. Math in Catscript is normal, and follows normal conventions of mathematics.

#### Null Value

The null value in Catscript allows a user to create a variable or point to another piece of data that has no value. This is done by using the `null` keyword.

## Creating and Using Variables

Variables are made and used in Catscript by using a keyword, followed by a name and value. These variables can then be accessed later by calling the name assigned during creation. Variables can be initialized with explicit or implicit typing. To explicitly declare a variable, use the following syntax: `var x : int = 0`. To implicitly declare a variable, use the following syntax: `var x = 0`. Ideally using implicit typing should not impose any disadvantage to explicit typing, but both options are provided to accommodate user preferences. It is worth noting that while variables are able to be declared implicitly, they cannot be assigned to new types. For example, an integer variable cannot be reassigned to become a string.

## Print statements

Print statements are implemented into Catscript as they are in most other programming languages by using `print(message)`. Print statements are also able to print variables and other data that might change, and is not limited to hardcoded values. It should be worth noting that print statements in Catscript will automatically add a new line after the printed message.

## String Concatenation

Catscript also supports string concatenation in the form of `"value" + "value"`. This can be used to combine strings, or to combine other data types into strings. For example, `1+2` will result in a value of 3 if used with arithmetic, but if the expression is instead `1+"2"`, the result will be 12, as 1 and 2 are concatenated together, rather than added.

## Comments

Catscript has commenting built into its syntax, which can be used with `//`. Comments in catscript ignore anything after the comment token until the end of a line. If a comment is placed in the middle of a line of code, any legitimate code in that same line will also be ignored.

## If Statements

If statements follow a standard format of `if(condition){result}`. These conditions can be comparisons of equality, which includes mathematical equality as well as checking object data, such as type and value.

For example:

`if(1+1==2)` is a valid condition. If a variable `x` exists, `if(x==2)` is also a valid condition, but if `x` has not been initialized, the if statement will not be valid.

## Comparisons and Equality

Comparisons in Catscript are standard and have no unique rules. Equalities are written as follows:

- Equals: `==`
- Greater Than: `>`
- Less Than: `<`
- Greater or Equals: `>=`
- Less Than or Equals: `<=`
- Not Equals: `!=`

## Boolean Expressions in Catscript

Catscript boolean expressions are simple to use, and consist of `true` and `false`. Like most other programming languages, the use of `!` or `not` can be used to nullify or reverse a boolean expression or condition. For example, `if(1>2)` will equate to `false`, but `if(!(1>2))` will equate to `true`. Similarly, `!true` will be `false`, and `!false` will be `true`.

## For loops

For loops in Catscript are implemented similarly to `if` statements, and is made up of the condition component as well as the body. A key difference between the `if` statement and the `for` loop is that the `for` loop must have its condition be built up of `x in y`, where `y` is usually a list. An example of a `for` loop is as follows:

```
for (x in [1,5,10]) {  
  print(x) //this will print 1, 5, 10 with new lines between each number.  
}
```

This `for` loop will iterate through the list of `[1,5,10]` and print out each value, similar to a `for each` loop in other languages. `For` loops in Catscript can also be nested, meaning multiple loops can run inside of each other and access the lists in loops of higher order.

## Functions

Functions can be declared and called in Catscript. To declare a function, use the `function` keyword followed by a name and set of parameters. Calling a function only requires the name and arguments to be passed in.

An example of defining and calling a function is as follows:

```
function func(x : int) {  
  return(x)  
}  
  
print(func(2)) //this will print '2'
```

This function simply returns a given input, meaning you can use it inside of a `print` statement to print a given value. For example, `print(func(5))` will print `'5'`. Functions in Catscript support recursion, and can reference themselves as many times as a program demands.

## Lists

A list in Catscript is a special data structure that is designed to store multiple entries of data. A list can store any data type that other structures can store, but lists cannot be changed after being initialized. A list in catscript appears as: `[index0,index1,index2,...]` and is accessed by calling the respective index for a given list. Lists can also be iterated through with `for` loops.

## Error Parsing

Catscript is designed with an error parser, and should alert a user of improper syntax and exceptions during compile time.

## Section 5: UML

<https://imgur.com/a/VHEIudy>

## Section 6: Design trade-offs

A design tradeoff made in this project is the use of recursive decent. Recursive decent has been a great way to implement the language because of its recursive nature. It's also very a useful way to learn. It's also more likely or us to see recursive decent in the future compared to something like a parser generator. Although I'm happy to have used recursive decent, a tradeoff to be mentioned is that a parser generator is more of an industry standard, and it requires less code.

## Section 7: Software development life cycle model

Test Driven Development (TDD) was used for this project, and I believe it is the best dev life cycle model I've used throughout my entire college career. I cannot think of a single hindrance it has caused me on this project. I believe every class that TDD would be applicable should be using it in some way. My highest ranking classes throughout 4 years have used TDD.