

[jaygforbes](#) / [csci-468-spring2022-private](#) Private[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)

master ▾

[csci-468-spring2022-private](#) / [capstone](#) /
Capstone.md[Go to file](#)

...

**jaygforbes** Update Capstone.mdLatest commit c5acead 6 minutes ago [History](#)

1 contributor

225 lines (156 sloc) | 9.91 KB



Raw

Blame



Section 1: Program

/capstone/portfolio/src.zip

Section 2: Teamwork

Team member 1: Designed most functions and also wrote nearly all the code for the project. Team member 2: Wrote 3 test cases for the code and Catscript documentation.

Team member 1: 130 hours were spent in development for this project. Team member 2: 1 hour was spend in the testing for this project, 3 Hours for Catscript documentation.

Section 3: Design pattern

Memoization/Flywheel was the design pattern that was used for this project. If you navigate to Parser -> CatscriptType.Java, on line 37 is where it begins. The hash map we created holds the cached objects. The if/else statement handles if the object has been cached. When executing this program, many objects are created, sometimes this process is repeated several times. Having memoization allows me to cache these certain events and creates far less data to be managed by memory while at the same time making the program more efficient.

Section 4: Technical writing.

Catscript Guide

Catscript is a coding language developed for the CSCI 468 Compilers capstone course at Montana State University - Bozeman. It is compiled into Assembly in the Java Runtime Environment and features a number of qualities typically associated with high-level programming languages.

The compiler for Catscript has been built with a recursive-descent parser, as compared to other more ubiquitous parsing methodologies.

Introduction

Catscript is a simple, but robust scripting language. It can perform a variety of tasks from basic mathematic operations, simple string manipulation, and complex control flows. Below is a rudimentary example of the language in action:

```
function standUpAndCheer(team : string) : void {  
    print("Go, " + team + ", Go!")  
}  
  
standUpAndCheer("Cats") // Go, Cats, Go!
```

Grammar and General Syntax

Primitive Types

There are three primitive types supported by Catscript at this time: `int` , `bool` , and `void` .

- Integer values, denoted with the reserved word `int` , are numeric values which contain no decimals. All integers in Catscript are signed integers and support a number of mathematical operations.

```
var x : int = 3
var y : int = -9
var z : int = 42
```

An example of several integer variable declarations

- Boolean values, denoted with the reserved word `bool` , are singular values which are like a switch, and can be either 'on' (`true`), or 'off' (`false`).

```
var p : bool = true
var q : bool = false
```

An example of two boolean variable declarations

- `void` types are exclusively used when declaring functions that do not return any information. Although the `void` type is only used for function declarations, the other two primitives may also be used as return values in function declarations.

```
function foobar() : void {}
```

An example of a function declarations with a void return type

Objects

In addition to the primitive data types, Catscript makes use of three types of object data types as well: `string`, `list`, and `null` objects.

- String values contain text. Unlike some higher-languages, the number of characters in a Catscript string does not need to be defined before assignment, and can be changed as needed.

```
var hello : string = "Hello World!"
```

An example of a string object

- List objects are collections of different values of the same type. The elements of a list object can either be primitive values or other objects. Similar to strings in Catscript, the size of a list object need not be defined during instantiation.

```
var list1 : list<int> = [1, 2, 3]
var list2 : list<string> = ["a", "b", "c"]
```

Two examples of a list object, one containing integer values, and the other containing string objects

- `null` is a value used to denote an object as having been undefined, and can be used to declare objects before assigning values to them.

Objects may also be declared using a generic `object` type, which allows a variable to have any type of object assigned to it during and after it is declared.

Variable Declaration and Assignment

Variables can be declared in Catscript using either hard or soft typing. However, all variables in Catscript must be instantiated upon declaration.

```
var x = "foo"  
var y : string = "bar"
```

Above is an example of soft/implicit typing (`var x = "foo"`), followed by an example of hard/explicit typing (`var y : string = "bar"`). Catscript variables can also be re-assigned after instantiation:

```
var x = "foo"  
x = "bar"
```

Even though Catscript variables may be re-assigned after being declared, their type cannot be modified (e.g., a variable instantiated as a string may not be assigned a boolean value). This includes variables declared using implicit typing, as the type of the variable is inferred from the initial value it is instantiated with.

Comments

Comments are a method of adding important or supplemental information to a Catscript file without changing its behavior. They are created by typing `//` anywhere in your script. Once done, all text after the two slashes will have no effect on the behavior of your script.

Comments can either be created on their own line, or inline with a piece of code:

```
// This comment has a line all to itself...  
  
var x = "no comment"    // ...and this comment comes after a  
variable declaration!
```

Control Flow in Catscript

Catscript's control flow functionalities come in three forms-- for loops, if statements, and functions. Each has their own use-cases which make them extremely valuable for a variety of different scenarios.

Each control block is able to encapsulate other sets of statements and expressions using the `{` and `}` symbols.

For loops

For loops are used to iterate over each value in a list, and are great for performing repeatable tasks.

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

For loops can also be nested to perform operations involving multiple lists at once, or multi-dimensional lists.

```
var matrix = [[3, 2, 3], [2, 3, 2], [2, 1, 2]]  
for (row in matrix) {  
  for (element in row) {  
    print(element)  
  }  
}
```

If Statements

If statements allow for conditional control of code operation. The most basic utility of if statements is to execute a set of commands which are predicated on another value.

```
var wordOfTheDay = "Serendipity";
```

```
if (wordOfTheDay != null) {  
    print("The word of the day is " + wordOfTheDay + "!")  
}
```

If statements also have a second form as if-else statements, which allow for optional commands which are executed exclusively based on a condition.

```
var wordOfTheDay = "Serendipity";  
if (wordOfTheDay != null) {  
    print("The word of the day is " + wordOfTheDay + "!")  
} else {  
    print("We're sorry, the word of the day appears to be  
missing.")  
}
```

Function Declaration

Functions are easily the most versatile tool for control flow in Catscript. They allow for sets of statements to be defined as single elements that can be executed in multiple parts of a script.

```
function weather() : void {  
    print("It's a sunny day today.")  
    print("How lovely!")  
}
```

```
weather()    // This line causes the above two print statements to  
be executed
```

Functions can also have a list of arguments, called parameters, passed to them. Those parameters can then be used to perform dynamic actions.

```
function weather(quality : string, feeling : string) : void {  
    print("It's a " + quality + " day today.")  
    print("How " + feeling + "!")  
}
```

```
}

weather("sunny", "lovely")      // "It's a sunny day today.",
                                "How lovely!"
weather("rainy", "dreadful")    // "It's a rainy day today.",
                                "How dreadful!"
```

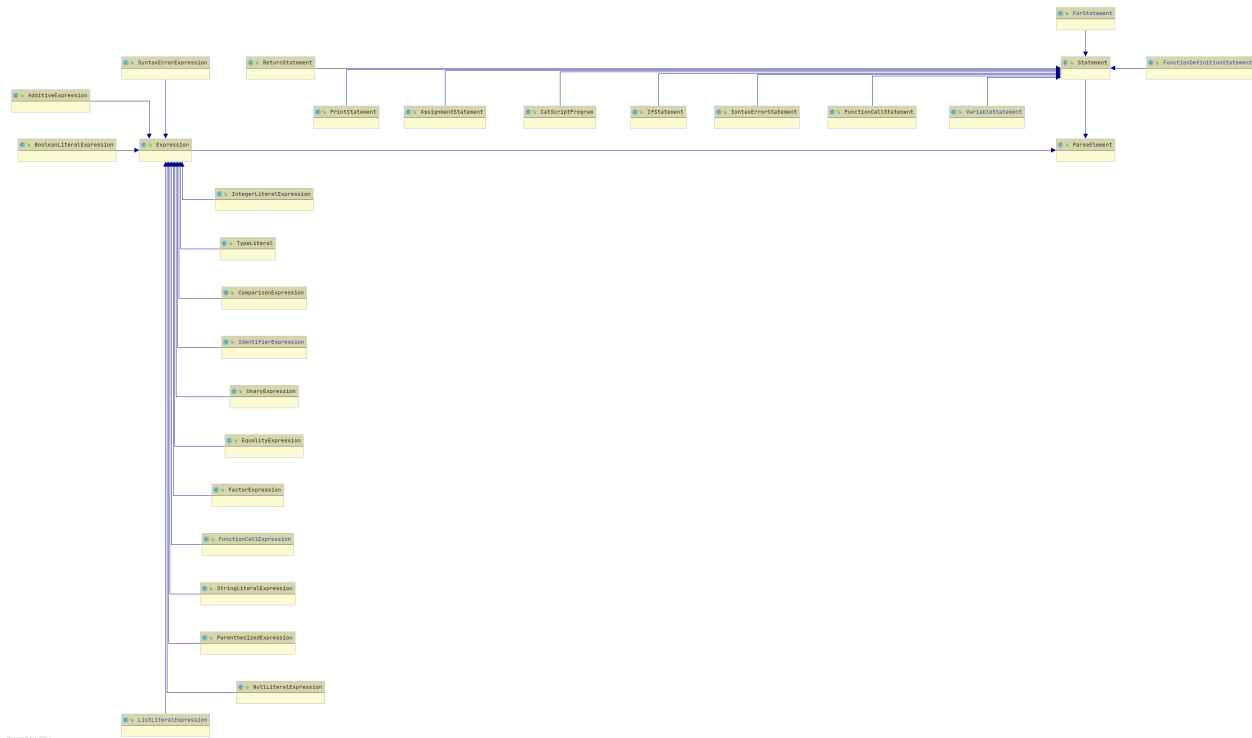
Finally, functions can also be executed recursively. That is, a function can call itself. This is an extremely powerful functionality, but can also cause your scripts to become unresponsive if done incorrectly, so be careful!

```
function go(count : int) : void {
    if (count > 0) {
        print(count * count)
        go(count - 1)
    }
}

go(5)    // output: 25    16    9    4    1
```

Control blocks can also be used in combination with one-another to perform complex operations. This includes functions which contain for loops and if statements, for loops that conditionally call functions, and more! The sky is the limit.

Section 5: UML for Catscript project.



Section 6: Design trade-offs

Recursive Decent vs Parse Generator We are using recursive descent parsing for this project. This is not the common method used for classes/projects like this, usually a parse generator is used. Using recursive descent does not compete with the parse generator in terms of speed. Parse generator is quicker in that regards. However, when learning parsing, the recursive descent parser is more intuitive and simple for beginners, allowing for a great learning experience. This is because recursive descent is a top-down method of implementing parsing that uses recursion and trees. When using context free grammar to implement the parser it was very simple because the context free grammar was very similar to the implementation of the code.

We did not use the visitor pattern. We put evaluation, compile, and everything else directly on the parse tree nodes. This makes things far more simple to learn for students who are new to compilers.

Section 7: Software development life

cycle model

We are using Test Driven Development (TDD) for this project. I found this incredibly helpful because it clearly lays out goals in a way that makes sense when you are actually looking at your IDE and code rather than a conceptual map. Having TDD is extremely helpful not only in the way mentioned before but in teamwork and time management settings. You can lay out how many tests you need to pass after a certain interval of time to stay on pace. In addition, with having several tests, it is easy to assign work to coworkers and reassign tests if someone were to fall behind in their work.