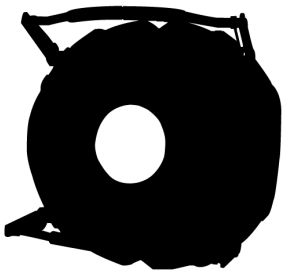
Wheatley



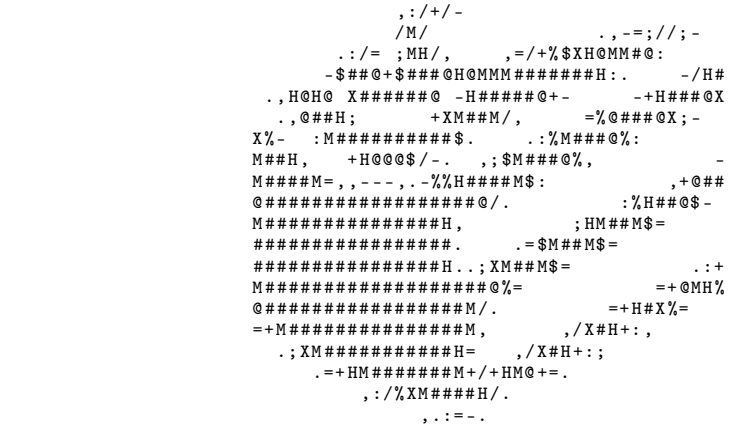
Weak Headed Evaluation in an Advanced Terminal Language Execution Y-environment

Language Summary and Example Programs

Michael Durso

michael.durso1@marist.edu

December 11, 2019



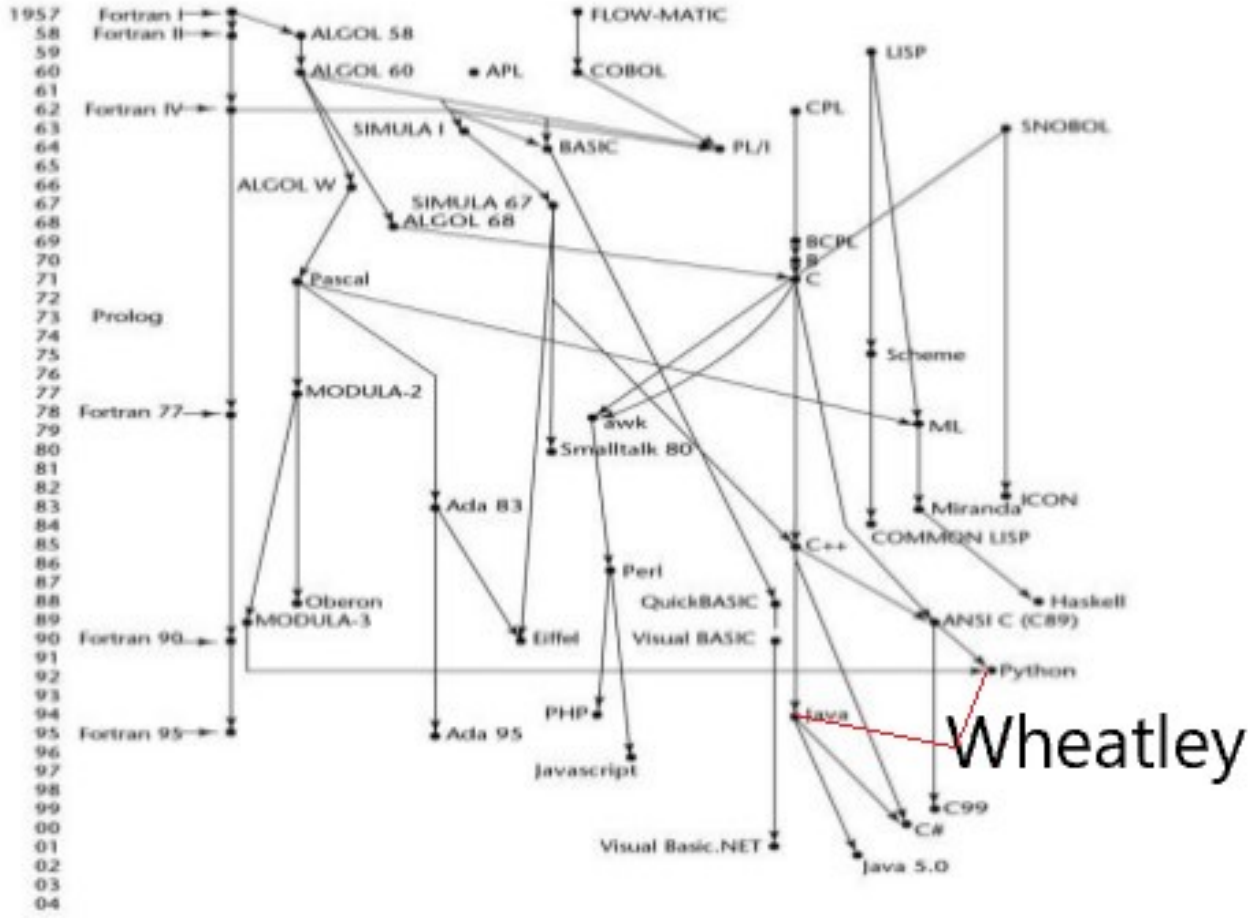
Contents

1 Introduction

Wheatley is or "Weak Headed Evaluation in an Advanced Terminal Language Execution Y-environment" is a simple language (even for the weak minded) as well as modern, strongly-typed, object-oriented language that works best in an Aperture Lab environment on the GLaDOS or "Genetic Lifeform and Disk Operating System" in a procedural environment. It is based on Java, Python, and most importantly Cake, but differing in the following ways:

1. Semicolons are not required for single line statements but are recommended for readability and debugging. If there are multiple lines inside an "if" statement then each line will require a semicolon.
2. The Brackets have been replaced so that "" is now "<3" and "" is now "</3".
3. Wheatley is strongly typed and statically scoped (utilizes early binding).
4. Null values are replaced with lie values. The cake is a lie.
5. To print you throw lemoNADES into your code. Print and println are: lemonadeOut and lemonadeOutln respectively.
6. Wheatley is compiled.
7. If you don't write in this language "...its not out of the question that you might have a very minor case of serious brain damage." -Wheatley

1.1 Genealogy

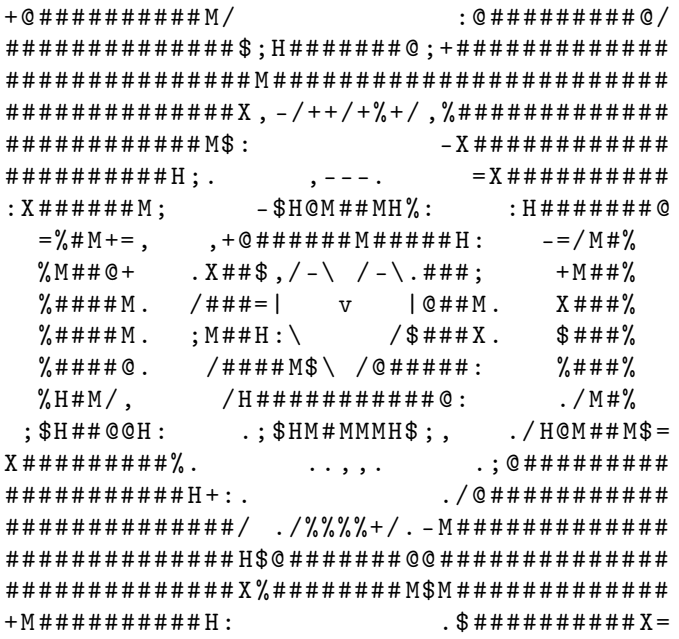


*The Rise of Wheatley came from a new language that would be optimal in creating a handheld portal device. The language is used in designing anything that is part of the GLaDOS. Cave Johnson put his "best" engineers (the only ones that were **still alive**) on the job to create this language out of fear for their house being burnt down by a combustible lemon. Before Wheatley was used, programmers would use either Java or Python so they took the best of everything and jammed it into one language. Java originated from C++ and its ancestors. Python also had some C++ in it as well as LISP and MODULA.*

1.2 Hello World

Commencing fulfillment of moral obligation to write the "Hello World" program.

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Hello World                                            <)
(>__________________________________________________________________<)

chapter MyFirstProgram
<3
  testChamber helloWorld
  <3
    course helloWorld()
      <3
        lemonadeOut("Hello World!");
      </3
  </3
</3
```

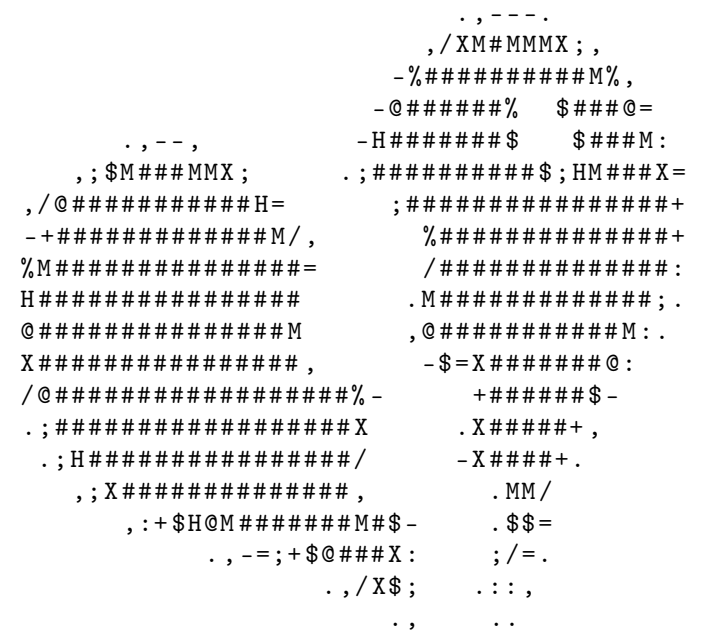


1.3 Program Structure

1.3.1 Organizational Concepts

The key organizational concepts in Wheatley are as follows:

1. Every testChamber (class) needs to be inside of a chapter (similar to a Java package or project). The chapter needs to have one or more testChambers in order for it to be a valid program.
2. All testChambers (classes) are public since every test is required to only potentially grant your freedom.
3. The atlas and pbody (getters and setters) are created using prefixes atlas (get) and pbody (set). It is then followed by the attributes name. Attributes are required to begin with a capital letter and best practice is to capitalize the first letter of every word in the attribute name.
4. Boolean values are pass and fail. You might get cake if you pass.
5. To enforce good style, white space is used similar to that of Python. Tabs or spaces can be used because all white space is converted to the space character when compiled.
6. Courses (Functions) are created similar to Python and Java but uses required arguments with the data type and value. See section 1.5 for a course declaration statement
7. Brackets from Java are replaced with <3 to open and </3 to close. You can essentially do a "replace all" from a Java program to have these in the correct location. It is a full heart and a broken heart referencing the line "Even though you broke my heart and killed me" from the song "Still Alive" which plays in the end credits of *Portal: Still Alive*.
8. A main function is not required.



1.3.2 Sample Program

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Pass the Level                                         <)
(>------------------------------------------------------------------<)

chapter PassTheLevel
<3
  testChamber level
  <3
    (> Default player
    course player(String Name, Boolean StillAlive, Boolean PuzzlesDone)
    <3
        subject.atlasName("");
        subject.atlasStillAlive("pass");
        subject.atlasPuzzlesDone("fail");
    </3

    (> Get the player information
    course getPlayer()
    <3
        subject.pbodyName();
        subject.pbodyStillAlive();
        subject.pbodyPuzzlesDone();
    </3

    (> Print the player information
    course printPlayer()
    <3
        lemonadeOutln(subject.pbodyName());
        lemonadeOutln(subject.pbodyStillAlive());
        lemonadeOutln(subject.pbodyPuzzlesDone());
    </3

    (> Check to see if level is passed
    course Boolean isLevelPassed()
    <3
     if (StillAlive == pass && PuzzlesDone == pass)
        return pass;
     else
        return fail;
    </3

~~0
newPlayer := Player("Alan", pass, pass);
newPlayer.printPlayer()
This would print:
Alan
pass
pass
0~~
  </3
</3
```

This creates a chapter (package) with a testChamber (class) that has four courses (functions). The first course is a constructor for the default player. Next there is a course to get the player information followed by a course to print that same information. Lastly there is a course to check if the player has finished their current level. This course demonstrates an if/else statement, checking Boolean values, and returning values. in the multi line comment there is a sample player with its output for printing the player. Shows how to use <3 and $</3$ in the correct manner. Shows how to use the comments.

1.4 Types and Variables

There are two kinds of types in Wheatley: ***value types*** and ***reference types***. Variables of value types directly contain their data whereas variables of reference types store references to their data, the latter being known as objects. With reference types, it is possible for two variables to reference the same object and thus possible for operations on one variable to affect the object referenced by the other variable.

1.5 Statements Differing from Java and Python

Statement	**Example**

if / else statements	course Boolean isLevelPassed()
	<3
	if (StillAlive == pass && PuzzlesDone == pass)
	return pass;
	else
	return fail;
	</3

Define an Array (rules)	num rules MyRules = [1, 2, 3, 4];
Define a Dictionary (instructions)	num instructions MyInstructions = <3 'light blue' 1, 'dark blue' 2, 'red' 3, 'orange ' 4, </3

Constructor	course player(String Name, Boolean StillAlive, Boolean PuzzlesDone)
	<3
	subject.atlasName("");
	subject.atlasStillAlive("pass");
	subject.atlasPuzzlesDone("fail");
	</3

Print and println	lemonadeOutln(“Hello World”); lemonadeOut(“Hello World”);

Method/Function declaration	course getPlayer()
	<3
	subject.pbodyName();
	subject.pbodyStillAlive();
	subject.pbodyPuzzlesDone();
	</3

2 Lexical Structure

2.1 Programs

A Wheatley ***program*** consists of one or more ***source files***. A source file is an ordered sequence of (probably Unicode) characters.

Conceptually speaking, a program is compiled using three steps:

1. Transformation, which converts a file from a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. Lexical analysis, which translates a stream of Unicode input characters into a stream of tokens.
3. Syntactic analysis, which translates the stream of tokens into executable code.

2.2 Grammars

This specification presents the syntax of the Wheatley programming language where it differs from Java and Python.

2.2.1 Lexical grammar where different from Java and Python

```
<Assignment operator>    -->  :=
<Mathematical operator>  -->  +  |  *  | /  | -
<Comparison operators>   -->  == |  != | <= | >=
<Keyword>                -->  <Language Defined>
                         -->  <Variable Defined>
<Begin block>            -->  <3
<End block>              -->  </3
<lemon>                  -->  a | b | c | ... | x | y | z
                         -->  A | B | C | ... | X | Y | Z
<num>                    -->  0 | 1 | 3 | ... | 7 | 8 | 9
<Single-Line comment>    -->  (>
<Delimited comment>      -->  ~~0 ... 0~~
```

2.2.2 Syntactic ("parse") grammar where different from Java and Python

```
<chapter declaration>     --> chapter <identifier>
<testChamber declaration> --> <access modifier> testChamber <identifier>
<function declaration>    --> <access modifier> <data type> <identifier> <parameter list>
                          --> <access modifier> <data type> <identifier>
<parameter list>          --> <parameter> <parameter list>
                          --> <parameter>
<parameter>               --> <data type> <identifier>
```

2.3 Lexical Analysis

2.3.1 Comments

Two forms of comments are supported: single-line comments and delimited comments. ***Single-line comments*** start with (> the characters and extend to the end of the source line. Depending on the font this may or may not look like the slice of cake it was intended to look like. ***Delimited comments*** start with the characters 0 and end with the characters 0 . These characters are supposed to be an object going in one portal and coming out the other simply skipping all of the content in between. Delimited comments may span multiple lines. Comments do not nest.

2.4 Tokens

There are several kinds of tokens identifiers, keywords, literals, operators, and punctuators. White space and comments are not tokens, though they act as separators for tokens where needed.

Wheatley Language Specification

tokens:

identifier

keyword

integer-literal

real-literal

character-literal

string-literal

operator-or-punctuator

2.4.1 Keywords different from Java or Python

A ***keyword*** is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier except when prefaced by the @ character.

New Keywords	Removed Keywords
pass	true
fail	false
lie	null
lemon	char
num	
Chapter	
testChamber	
course	
lemonadeOut	print
lemonadeOutln	println
	public
	private

3 Types

Wheatley types are divided into two main categories ***Value types*** and ***Reference types***.

3.1 Value Types (different from Java and Python)

Boolean- a value that can be either pass or fail, representing true and false.

Lie- Lie is a null type representing no value as well as no cake.

Lemon- works the same as a char.

Num- a number type that automatically casts between ints, doubles, floats etc. Can be less accurate if misused. Other number types are still available and work the same.

3.2 Reference Types (differing from Java and Python)

rules- a systematic arrangement of data (an array).
Examples:

num rules MyRules := [1, 1.2, 1.34, 35];

lemon rules MyRules := ['a', 'b', 'c', 'd'];

String rules MyRules := ["fatty", "fatty", "no", "parents", "-Wheatley"];

multiType rules MyRules := [1, 'a', "GLaDOS"];

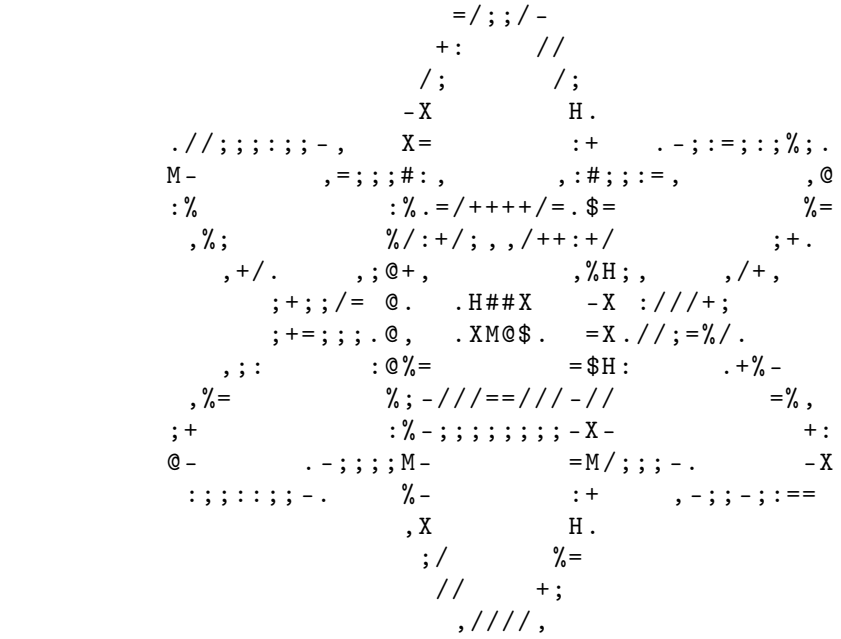
instructions- a collection of unordered values accessed by key rather than by index.

```
num instructions MyInstructions :=
  <3
   'light blue'  1,
   'dark blue'   2,
   'red'         3,
   'orange'      4
  </3
```

4 Example Programs

4.1 Program Example 1: Caesar Cipher Encrypt

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Caesar Cipher Encrypt and Decrypt                      <)
(>__________________________________________________________________<)
chapter Caesar
<3
    testChamber CaesarCipher
    <3
        (> encrypt function
        course String encrypt(String Phrase, num ShiftAmountt)
        <3
            Portal := "";
            for I through range(len(Phrase))
            <3
                lemon := Phrase[I]
                if (lemon.isUpper())
                  Portal += chr((ord(lemon) + ShiftAmountt - 65) % 26 + 65);
                else
                  Portal += chr((ord(lemon) + ShiftAmountt - 97) % 26 + 97);
                return Portal;
            </3
        </3

        (> decrypt function
        course String decrypt(String Phrase, num ShiftAmount)
        <3
            Portal := encrypt(Phrase, ShiftAmountt * - 1);
            return Portal;
        </3
    </3
</3
```



4.2 Program Example 2: Caesar Cipher Decrypt

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Caesar Cipher Encrypt and Decrypt                      <)
(>------------------------------------------------------------------<)

chapter Caesar
<3
    testChamber CaesarCipher
    <3
        (> encrypt function
        course String encrypt(String Phrase, num ShiftAmountt)
        <3
            Portal := "";
            for I through range(len(Phrase))
            <3
                lemon := Phrase[I]
                if (lemon.isUpper())
                  Portal += chr((ord(lemon) + ShiftAmountt - 65) % 26 + 65);
                else
                  Portal += chr((ord(lemon) + ShiftAmountt - 97) % 26 + 97);
                return Portal;
            </3
        </3

        (> decrypt function
        course String decrypt(String Phrase, num ShiftAmount)
        <3
            Portal := encrypt(Phrase, ShiftAmountt * - 1);
            return Portal;
        </3
    </3
</3
```

4.3 Program Example 3: Factorial

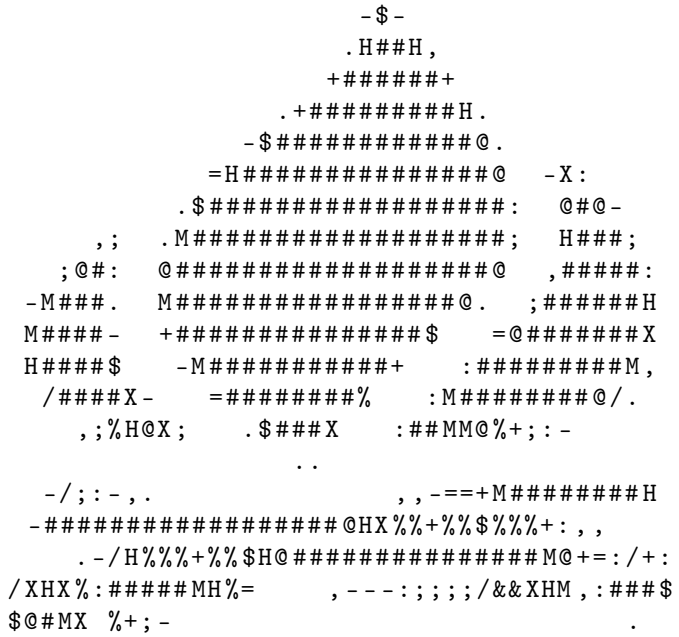
```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Factorial                                              <)
(>------------------------------------------------------------------<)

chapter Factorial
<3
    testChamber Factorial
    <3
        (> recursive factorial function
        course num factorial(num I)
        <3
            if I == 0
              return 1;
            else
              return I * factorial(I - 1);
        </3
    </3
</3
```

4.4 Program Example 4: Insertion Sort

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Insertion Sort                                         <)
(>------------------------------------------------------------------<)

chapter Insertion
<3
    testChamber InsertionSort
    <3
        num rules Portal := [343, 117, 2112, 007, 8675309, 42];

        course swap(num I, num J)
        <3
            num Temp := Portal[J];
            Portal[J] := Portal[I];
            Portal[I] := Temp;
        </3
        course insertionSort(num[] Portal)
        {
            for I through range(I)
            {
              num J := I;
              while ((J > 0) && (Portal[J] < Portal[J - 1]))
              {
                swap(J, J - 1);
                J := J - 1;
                lemonadeOutln(rules.toString(portal));
              }
            }
        }
    </3
</3
```

4.5 Program Example 5: Bubble Sort

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Bubble Sort                                            <)
(>__________________________________________________________________<)


chapter Bubble
<3
    testChamber BubbleSort
    <3
        course rules BubbleSort(rules Portal)
        <3
            for I through range(len(Portal) -1, 0, -1)
            <3
                for J through range(I)
                <3
                    if Portal[I] > Portal [I + 1]
                    <3
                      Holder := Portal[I];
                      Portal[I] := Portal[I + 1];
                      Portal[I + 1] := Holder;
                    </3
                </3
            </3
        </3
    </3
</3

~~0
num rules Portal := [343, 117, 2112, 007, 8675309, 42];
BubbleSort(Portal);
lemonadeOut(Portal);
0~~
```



4.6 Program Example 6: Fibonacci Sequence

```
(> Michael Durso                                                    <)
(> Professor Labouseur                                              <)
(> CMPT331 - Theory of Programming Languages                        <)
(> Wheatley: Fibonacci Sequence                                     <)
(>-----------------------------------------------------------------<)

chapter Fib
<3
    testChamber Fibonacci
    <3
        num PrintAmount := 10;
        num Val1 := 0;
        num Val2 := 1;
        lemonadeOut("Fibonacci Sequence of "+count+" numbers:");

        num I := 1;
        while(I <= PrintAmount)
        <3
            lemonadeOut(Val1 + " ");
            num sumOfPrevTwo = Val1 + Val2;
            Val1 = Val2;
            Val2 = sumOfPrevTwo;
            I++;
        </3
    </3
</3
```