

CS 315 - Programming Languages

Project Part 1

*Fall 2016*

APES: a Language that Draws Shapes

Group 1-7

Abdullah Al-Wali

Selin Özdaş

Utku Uçkun

**Table of Contents**

[**Regular Expressions**](#_9bcusx7vaoz3) **3**

[**Types I: Primitive Types & their Operations**](#_p5ynloqisild) **3**

[Integer & Float](#_69k9yny6419d) 3

[Boolean](#_d9662htomy2) 4

[String](#_f19en5fq7rro) 4

[**Types II: Location, Size, and Colour**](#_isrs74oht8wb) **5**

[Location](#_h7uiahbw21be) 5

[Size](#_is468sao5cw5) 5

[Colour](#_z2m2ntnotncp) 5

[**Types III: Shapes**](#_4h60lrofjjo4) **6**

[Initializing Shapes](#_4y67szbuho5u) 6

[Line Type](#_vkx5n4qn678) 7

[Rectangle Type](#_onds76b78ws3) 7

[Oval Type](#_w9pbzov69g85) 8

[String Shapes](#_5ybbuxvw4ejt) 8

[Strings can be represented as shapes in Apes and can become composite shapes by being merged with other shapes. A string shape takes string as an argument. Default value of this function is empty string and can be initialized by the following:](#_novq0m58zgfd) 8

[StringShape(String message = “”)](#_fp8qtkt2v7g5) 8

[Composite Shapes](#_84pgih93imx8) 8

[**Draw Function**](#_fb3up9fyo9sm) **8**

[**Loops**](#_l3w8qni19bjc) **9**

[**Conditional Statements**](#_oe241vrzgsg) **10**

[**Comments**](#_5lm5tnzh2wjm) **10**

# Regular Expressions

We will need some regular expressions on which the language will be built, some of these expressions are adapted from the notes or the lexing examples on the course website. All the expressions needed and used will be listed below:

DIGIT: [0-9]

CHARACTER: [A-Za-z]

MUL\_OP: \* | / | %

ADD\_OP: + | -

UNARY: -

IDENTIFIER: CHARACTER(CHARACTER | DIGIT)\*

**Boolean:** “true” | “false”

**Integer:** DIGIT+

**Float:** DIGIT\*(\.DIGIT|DIGIT\.)DIGIT\*

**String:** \"([^\"\\\n]|\\\"|\\n|\\t|\\\\)\*\"

The bold expressions refer to the 4 primitive types of our language

# Types I: Primitive Types & their Operations

There are 4 primitive types in our language: String, Integer, Float, and Boolean. The language is dynamically typed. Syntax details for creating and initializing variables are given in the tutorial.

## Integer & Float

The Integer and Float will support the standard arithmetic operators: +, -, /, \*, %

An operation between two Integers will result in an Integer

An operation between two Floats will result in a Float

An operation between an Integer and a Float will result in a Float

Float ← Float OP Float

Integer ← Integer OP Integer

Float ← Integer OP Float

Float ← Float OP Integer

**Note:** in our report we will use (Type ← ) to denote that an operation evaluates to a certain type in contrast to (Type →) which is used in CFG.

## Boolean

The following operators are supported with boolean types: && (and), || (or), and !(not). The equality operator, ==, will be used for checking equality of strings

Logical operations on booleans will always result in booleans:

Boolean ← Boolean LogicalOP Boolean

Where

LogicalOP ← &&

LogicalOP ← ||

LogicalOP ← !

## String

Strings support concatenation with other strings and with Integers and Float. Concatenation will be done using the ‘+’ operator. The result of concatenating a string with Integer or Float given that the String is on the left hand. Supported conversions are as follow:

String ← String + String

String ← String + Float

String ← String + Integer

Note that those below are not supported:

String ← Integer + String

String ← Float + String

# Types II: Location, Size, and Colour

## Location

The Location type has two mandatory properties, the coordinates x and y, which are of type Integer. Creating a Location variable would be as follows:

IDENTIFIER = Location( x , y);

Accessing x and y values will be done using the ‘.’ operator on the variable name. Location can also be created in place when used as a function parameter, this will be shown in the tutorial.

## Size

The Size functions in a similar way to the Location type. Instead of x and y, Size uses width and height as attributes. Both are of Integer type and can be accessed using the ‘.’ operator.

IDENTIFIER = Size( width, height);

## Colour

The Colour type will have three attributes: Red, Green, and Blue. Each of these attributes is an Integer ranging from 0 to 255. Creating a Colour type variable will be as follows:

### 

IDENTIFIER = Colour(red, green, blue);

Note that Colour is spelled with a “u”according to the authentic British spelling.

The Colour type will also have a constructor that takes a string, a colour name, and initializes the variable’s RGB values to that colour. The string must match one of the constants defined below, and must be typed in capital case.

|  |  |
| --- | --- |
| **Colour Name** | **RGB Value** |
| *BLACK* | (0,0,0) |
| *WHITE* | (255,255,255) |
| *RED* | (255,0,0) |
| *GREEN* | (0,255,0) |
| *BLUE* | (0,0,255) |
| *YELLOW* | (255,255,0) |
| *GREY* | (128,128,128) |

# 

# Types III: Shapes

## Initializing Shapes

Our drawing language supports some built-in shapes to make users drawing experience

better and easier. These pre-loaded shapes are ovals, rectangles and lines.

All shapes will follow this syntax for initialisation:

IDENTIFIER = ShapeName( *Parameters* );

Where

ShapeName → “Line” | “Oval” | “Rectangle” | “Composite” | “StringShape”

And *Parameters* means there will be some mandatory and some optional parameters depending of the shape we chose.

## Line Type

The Line shape has the following parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Default Value** |
| direction | String | “N” |
| startArrow | Boolean | false |
| endArrow | Boolean | false |
| arrowSize | Integer | 1 |

The constructor of a Line type has the following format:

Line( String direction = “N”, Boolean startArrow = false, Boolean endArrow = False, Integer arrowSize= 1)

Where

direction → “N” | “W” | “S” | “W” | “NW” | “NE” | “SW” | “SE”

All the parameters are optional and will be entered either in order or using named parameters.

Examples are provided in the tutorial.

## Rectangle Type

Rectangle has one optional parameter which allows it to have rounded corners.Default value of the degree of rounded corners is 0 (sharp corners). Therefore it has one constructor is as follows:

Rectangle( Integer roundCorners = 0 )

The value of roundCorners reflect how rounded the corners will be, ranging from 0 to 10.

## Oval Type

The Oval type will have no parameters. Therefore its constructor takes no parameters:

Oval()

## String Shapes

## Strings can be represented as shapes in Apes and can become composite shapes by being merged with other shapes. A string shape takes string as an argument. Default value of this function is empty string and can be initialized by the following:

### StringShape(String message = “”)

## Composite Shapes

Apes language allows user to create and use their own shapes.

Creating new shapes by using built-in shapes is available with the following function call:

Composite(String name, Shape shape1, Shape shape2)

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Default Value** |
| name | String | “newShape” |
| shape\_1 | Shape | - |
| shape\_2 | Shape | - |

Composite function takes four parameters to identify the newly created shape. First parameter is the name of the new composite. The second and third parameters are the shapes about to be composited. These parameters are not optional. Imagine the composite of two ovals with same size and same center. This will not be a new shape, it is going to be just another standard oval, so if their location and size are the same, the x parameter of the second shape is going to be increased by 1.

# Draw Function

The draw function will be used to draw the shapes with parameters passed to specify location, size, colour, etc.

The function will be called through the shape using the ‘.’ operator:

shapeName.draw(*parameters*)

The draw function has several parameters which will be detailed below. All parameters have been made optional and given default values in order to allow the user to draw a number of shapes free of scale (even the count can be optional and is defaulted to 1).

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Default Value** |
| location | Location | Location(0,0) |
| size | Size | Size(10,10) |
| strokeWidth | Integer | 1 |
| fillState | Boolean | False |
| fillColour | Colour | Colour(“Black”) |
| count | Integer | 1 |

The ordering of the parameters will be the same of that in the table, however, the function accepts named parameters in order to support optional parameters, and it is preferred that the user enters named parameters in the following format: *parameterName*: *value*

# Loops

Our language supports multiple iterations via defined loops. To enlarge the scale of preferences our language includes while, do while and for loops.

|  |  |
| --- | --- |
| **Loop Name** | **Loop Syntax** |
| while | while (statement) { ... } |
| do while | do {...} while ( statement ) |
| for | for( expression ; statement ; expression) { ... } |

boolean ← statement

expression ← IDENTIFIER OP expression | IDENTIFIER UNARY | UNARY IDENTIFIER | IDENTIFIER

***While loop:*** While loop is being executed repeatedly based on the given statement. While given statement is true according to its corresponding boolean parameter, code block inside curly braces will be executed. When given statement turns out to be false, loop execution will be aborted.

***Do while loop:***  Do while loop is a different version of while loop where the code block within curly braces is going to be executed at least once. After the first execution, it continues to execute the same code block unless the boolean definition of the statement becomes false.

***For loop:*** For loop is an extended version of while loop. The change is, the statement may be based on the changes in the control value. When any control value is not defined, it behaves like an ordinary while loop.

# Conditional Statements

|  |  |
| --- | --- |
| **Statement** | **Statement Syntax** |
| if | if ( statement ) {...} |
| else | else{...} |

If - else statements are also supported in Apes. If the statement turns out to be true, code block related to ‘if’ is going to be executed. If the statement is false, code block related to ‘else’ is going to be executed.

# Comments

Comments are supported in Apes. Comment syntax is the following:

/\* comment \*/

Inner comments are supported too.

/\* comment /\* comment \*/ comment \*/