Final Project: Compilers - II
Prof. Ramakrishna Upadrasta
Indian Institute of Technology, Hyderabad

Tangent

Language Specification

Team Members:

AMAN PANWAR - CS20BTECH11004

PRANAV K NAYAK - ES20BTECH11035

SHAMBHU KAVIR - CS20BTECH11045

SHREYA KUMAR - ES20BTECH11026

TAHA ADEEL MOHAMMED - CS20BTECH11052

VIKHYATH - CS20BTECH11056

Contents

INTRODUCTION TO TANGENT

1.1 Description

Tangent is a statically typed, object-oriented programming language intended for creating vector graphics. It takes design cues from Python's Matplotlib library and C++'s object-oriented functionality. Using built-in classes on an SVG, Tangent can be used to plot a variety of complicated figures, such as Bézier curves and polygons. Tangent also aims to offer users standard general-purpose programming language functionality such as looping and control flow. Tangent, while being object-oriented, aims to be simple, readable, and portable.

1.2 Motivation behind Tangent

Existing libraries that provide graph-plotting capabilities are hampered by the format that they encode images in, namely, raster graphics. For example, matplotlib produces its images as jpeg files, which, while having many benefits, cannot be relied upon for visual clarity. The very nature of the encoding in the form of bitmap files means the image is static, and zooming in will only expose the pixels whose states were set when the image was first generated.

SVGs take advantage of the nature of vector graphics by placing shapes in a 2D environment by way of a program consisting of a sequence of definitions for shapes and curves. This leads to a smaller file size while still retaining quality. Since the images are the result of a sequence of instructions, they can be re-rendered for every resolution, meaning there is no loss of quality when zooming in. This is especially useful when precision is required in images, like in the case of printing.

1.3 Design Goals

We aim for Tangent to be a user-friendly and particularly, a beginner-friendly language. Users should be able to create vector images with this language in a simple, elegant way. Beginners should be able to rapidly integrate this language into their projects because of the simplicity of the language itself, being very similar in structure to other C-style general-purpose languages.

Tangent's object-oriented facets include inheritance and polymorphism. Programming in Tangent is meant to be intuitive and convenient, allowing users to model real-world graphs and images with a level of control that normal bitmaps simply cannot allow, by harnessing their experience programming in other object-oriented languages towards creating images by making use of the built-in classes (e.g. line, curve, polygon, bezier, etc.) to build their own user-defined classes as per their needs or those of their clients.

LEXICAL CONVENTIONS

In order to retain similarity with other popular programming languages, the grammar for Tangent takes inspiration from that of C++. This allows for an easy transition between Tangent and other languages used in the user's project, making integration seamless. The scanning and parsing for the language's programs are done through flex, the lexical analyzer generator, and bison, the general purpose parser generator.

2.1 Comments

Comments are similar to those of the Python language, meaning they are indicated by a '#' at the start of each comment. They can only exist in a single line, and the compiler reads a newline ('\n') or an end-of-file (EOF) as the comment having been terminated.
Multiline comments can be thought of as a collection of single line comments.

2.2 Identifiers

Identifiers in Tangent can be strings of letters, numbers and underscores but they *must* start with a letter which can be followed by any number of alphanumeric characters and underscores. Whitespace is not allowed inside an identifier. In this, Tangent complies with the standard set by C and followed by C++, Java, and Python.

2.3 Keywords

The following table contains all the basic keywords :

`int`	`float`
`string`	`bool`
`void`	`var`
`family`	`if`
`else`	`for`
`while`	`switch`
`case`	`break`
`continue`	`true`
`false`	`send`
`const`	`me`
`public`	`private`
`Point`	`Path`
`Image`	`Rectangle`
`Circle`	`Ellipse`
`Polygon`	`Curve`
`func`	`Pi`
`Colour`	

The above listed keywords are case sensitive. Users are free to create identifiers as long as they differ from all the keywords by at least one character. The keywords themselves are reserved and cannot be used as identifiers.

2.4 Punctuations

- Commas (‘ , ’) are used to separate identifiers and constants in sequences during variable assignment, declaration and function argument list.
- All statements are terminated by semicolons (‘ ; ‘).
- Argument lists are enclosed by the following digraph: ‘<: :>’.
- Scope is defined by enclosing blocks in the following digraph: ‘(: :)’.
- Inheritance of family is mentioned by the following ‘::’.

2.5 Operators

Majority of the operators used in Tangent are similar to C++. The operators used are as follows:-

a. Arithmetic Operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Assignment	:=

b. Logical Operators:

Logical And	&

Logical Or	|
Logical Not	!

c. Relational Operators:

Equal To	=
Greater Than	>
Less Than	<
Greater Than or Equal To	>=
Less Than or Equal To	<=
Not Equal To	!=

d. Unary Operators:

Positive Indicator	+
Negative Indicator	-

2.6 Precedence and Associativity

Precedence follows the template set by languages like C and Java, with the precedence order being as follows:

Unary Operators	↓
Functions	
Parentheses	
Multiplication, Division and Modulo	

Addition and Subtraction	Precedence Decreasing downwards
Relational Operators	
Logical Operators	

2.7 Constants

Constants are values treated as is, i.e., literals. The following tables display some examples of various constants that can be used in Tangent:

- int constants:

3	-2	0

- floating point constants:

1.62	-0.456	123.456e-2

- string literals:

“Hello World!”	“a”	“new”

- bool constants:

true	false

2.8 Error Handling

By creating strong token definitions, we have attempted to maximise error handling at the lexical stage. Once the compiler encounters an error, the line number and associated error message should be displayed on the command line.

The following are examples of compilation errors

- when a rule evaluates to a false predicate.
- when some input does not match any expression.
- when none of the grammar rules provides a specific path to follow.

TYPES

3.1 Type Declaration

Data types are a collection of values with comparable properties. Primitive data types are those that the language pre-defines and are designated by a keyword. Fundamental data types are combined to create derived data types.

3.2 Primitive Types

Tangent takes inspiration for its primitive data types from other general purpose programming languages, like C++ and Java. Its primitive data types can handle integers, floating point numbers, boolean values, and strings.

Characters in Tangent are treated as strings of unit length.

3.3 Derived Types

Just like any ordinary language, arrays in Tangent can be used to store multiple data of the same type together. Multi-dimensional arrays can also be created which shall be implemented using a single array of larger size in the background. For example:

```
var int[5] example;
```

will declare an array of integers of size 5.

A ‘Point’ object defines a mathematical point on a 2D plane.

A ‘Curve’ object defines a mathematical 2D curve.

‘Rectangle’ represents a collection of 4 points that form a rectangle. It is defined using the attributes length, breadth and centre of the rectangle.

‘Ellipse’ represents a mathematical ellipse. Its attributes are length of major axis, length of minor axis and centre of the ellipse.

‘Circle’ similar to Ellipse, represents a mathematical circle. Its attributes are centre point and radius.

‘Polygon’ is a mathematical regular polygon defined by the number of sides, the size of each size and the centre of the polygon. We can use this to make equilateral triangles, squares and so on.

‘Colour’ is a pre-defined family which lets us define the rgb values for a colour. It helps us set the colour of an image.

STATEMENTS

All statements are terminated by semicolons (‘ `;` ‘).

4.1 Variable declaration

The variable declaration and assignment template is as follows:

$$\texttt{var <type> <identifier> [:= <value>]}_{\texttt{opt}}\texttt{;}$$

For example:

```
var float my_num;     var int your_num := -12;
```

4.2 Branching Statements

Tangent follows C style scoping for branches, with scoping being enforced to deal with the if-else ambiguity. Scope is, as mentioned in section 2.4, defined by the following digraph: ‘`(:   :)`’

- `if (: … :)`
- `else (: … :)`
- `switch (: … :)`
- `case`
- `break`
- `continue`

4.3 Loops

Loops are also scoped with ‘`(:  :)`’

- `while (: … :)`
- `for (: … :)`

OBJECT ORIENTED PROGRAMMING WITH TANGENT

We knew from the beginning that the design of our language would be object-oriented, allowing us to provide the user all the resources they need to expand on already-existing data types and create complicated designs that could be implemented using bezier curves to get the desired results. Some of the most fundamental and practical object-oriented programming ideas are listed below, along with how our language handles them:

5.1 Abstraction

An OOP language's objects offer an abstraction that conceals the internal implementation specifics. Similar to the coffee maker in your kitchen, all you need to know to do a certain action is which methods of the object are accessible to call and which input parameters are required while the actual implementation of something might not be necessary to know.

So, when it comes to abstraction in Tangent, there are some functions that can help the user with some functions that might be complex otherwise. For example, if a user would like to rotate the rectangle by a particular angle or say if a user would like to change the centreThe goal of the programming language Tangent is to be user-friendly for beginners. Users may create vector pictures with this programme in a simple, stylish way. Beginners may rapidly come up to speed with the language because of how it is designed.The goal of the programming language Tangent is to be user-friendly for beginners. Users may create vector pictures with this programme in a simple, stylish way. Beginners may rapidly come up to speed with the language because of how it is designed.The goal of the programming language Tangent is to be user-friendly for beginners. Users may create vector pictures with this programme in a simple, stylish way. Beginners may rapidly come up to speed with the language because of how it is designed.The goal of the programming language Tangent is to be user-friendly for beginners. Users may create vector pictures with this programme in a simple, stylish way. Beginners may rapidly come up to speed with the language because of how it is designed. point of a circle around the image, the user could call an existing method to do the same and not worry about how that particular method gets the work done. However, the user may utilize the needed functionality by following the general blueprint that the class contains.

5.2 Encapsulation

Encapsulation is the term used to describe the packaging of data and information into a single unit. Encapsulation is the binding together of the data and the functions that manipulate them in object-oriented programming. Our language requires this feature to make sure that users don't attempt to modify particular characteristics defined in classes. No function from the class may be accessed directly. To access the method that uses the class member variable, we require an object. Encapsulation refers to the usage of all member variables by the function that we create inside the class.

5.3 Inheritance

Inheritance allows us to derive from other classes, ‘inheriting’ their member variables and functions. Hence we can reuse code and better represent the relationships between classes. Inheritance is crucial for ‘Tangent’, as the different classes representing graphical objects have many “is-a” relationships, such as squares & rectangles, ellipses & circles, or user defined classes. The syntax for inheritance is similar to C++.

5.4 Polymorphism

Polymorphism is a core concept in OOPS that allows us to describe situations in which something occurs in several different forms. Mainly, we support function and operator overloading, allowing us to have several functions of the same name but with different parameters. A basic example of this in our language is overloading of Image.draw(), which can take various different types as a parameter, such as points, rectangles, curves, polygon, etc;

SAMPLE PROGRAMS

Example 1

```
family point(:
      var float x;
      var float y;
:)
family rectangle (:
      var point a;
      var point b;
      var point c;
      var point d;
:)
family square :: public rectangle (:
      var point side;
:)

#func <type> <function_name> <: <type> arg1, <type> arg2 :>(:
#     var <type> v1 := <value> + arg1;
#     send v1;
#:)
```

Example 2:

```
family vehicle(:
      var int wheels;
public:
      func int get_no_of_wheels<: :>
      (:
            send me >> wheels;
      :)
:)
family car :: vehicle(:
      var Colour paint_colour;
public:
      car<: :>(:
            me >> paint_colour := get_color_from_rgb<:10,10,10:>;
      :)
:)
main<: :>(:
      var car my_car;
      print(my_car >> get_no_of_wheels<::>);
      #first figure
      var Image I;
      I.set_dimension<:100,100:>;
      var Point A;
      A := make_point<:10,30:>;
      I.draw<:A:>;
      Path P;
      P.add_point<:A:>;
      P.add_point<:A + make_point<:0,20:>:>;
      I.draw<:P:>;
      I.output<:"my_file.svg":>;
:)
```