

# 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

<b>1. Introduction To Tangent</b>	<b>4</b>
1.1 Description	4
1.2 Motivation behind Tangent	4
1.3 Design Goals	5
<b>2. Lexical Conventions</b>	<b>6</b>
2.1 Comments	6
2.2 Identifiers	6
2.3 Keywords	6
2.4 Punctuations	7
2.5 Operators	8
2.6 Precedence and Associativity	9
2.7 Constants	9
2.8 Error Handling	10
<b>3. Types</b>	<b>11</b>
3.1 Type Declaration	11
3.2 Primitive Types	11
3.3 Derived Types	11
<b>4. Statements</b>	<b>13</b>
4.1 Variable Declaration	13
4.2 Branching Statements	13
4.3 Loops	13

---

<b>5. Object Oriented Programming with Tangent</b>	<b>14</b>
5.1 Abstraction	14
5.2 Encapsulation	15
5.3 Inheritance	15
5.4 Polymorphism	16
<b>6. Sample Programs</b>	<b>17</b>

---

# 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 and C++'s functionality. Using built-in classes on an SVG, Tangent may be used to plot a variety of complicated figures, such as bezier curves and polygons. Tangent offers the capability to carry out standard programming activities as well as the simplicity of charting and viewing two-dimensional data. Despite Tangent being an object oriented language, we are trying to make it simple, readable, and portable.

## 1.2 Motivation behind Tangent

The industry's existing libraries provide the necessary graphs but sacrifice visual clarity in order to scale. For instance, the readability of the jpeg files produced by *matplotlib* is hampered by the fact that when they are zoomed in, their clarity deteriorates. Tangent comes into play in this situation. Tangent creates vector images as SVG files that are independent of pixel values, thus quality is preserved regardless of how much an image is expanded or compressed. Additionally, Tangent allows users to mathematically define what constitutes the image, this is useful in scenarios where you need precision in the images.

## 1.3 Design Goals

The goal of the programming language Tangent is to be user-friendly for beginners. Users may create vector images with this programme in a simple, elegant way. Beginners may rapidly come up to speed with the language because of how it is designed.

---

Tangent aims to give users tools for object-oriented programming and data representation. This is accomplished by including interface mechanisms and object-oriented class methods into the design process. As a result, Tangent is modular and the code is reusable. The user also has the option to implement classes and objects exactly like in any other conventional object-oriented language by building on fundamental classes like squares and circles to produce sophisticated shapes and plots with the help of bezier curves and polygons.

Tangent is a multifaceted tool with Inheritance and Polymorphism. It also provides the way for the development of an intuitive approach to model real-world graphs and images, which is what our team believes is the future of programming. User defined classes can have other data attributes and methods just like in conventional languages such as C++.

---

## LEXICAL CONVENTIONS

The grammar for the lexical analyzer will be written using flex while trying to maintain a close resemblance to standard language specifications like that of C++ so that user can transition easily between Tangent and other standard programming languages because if otherwise, users would have to learn an entirely different syntax while making minor adjustments and improvements to make it better and visually appealing.

### 2.1 Comments

Comments are single-line only, and are indicated by a '#' at the start of each comment. A comment ends when a newline ('\n') or EOF is encountered. Multiline comments can be implemented by prefacing each line with a pound sign. This is similar to how comments are implemented in Python language.

### 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. We are trying to maintain the global standard notation used in major programming languages such as C++, Python, JAVA to let the user maintain their current practices while switching to Tangent.

### 2.3 Keywords

The following table contains all the basic keywords :

---

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

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:

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

b. Logical Operators:

<b>Logical And</b>	<b>&amp;</b>
<b>Logical Or</b>	<b> </b>
<b>Logical Not</b>	<b>!</b>

c. Relational Operators:

<b>Equal To</b>	<b>=</b>
<b>Greater Than</b>	<b>&gt;</b>



---


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	 Precedence Decreasing downwards
Functions	
Parentheses	
Multiplication, Division and Modulo	
Addition and Subtraction	
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 maximize error handling at the lexical stage. The lexical analyzer will totally disregard comments. At the conclusion of the compilation of the code, line numbers and the associated errors will all be shown at once (possibly on command

---

line interface). One of the following situations will result in a syntax error being recorded:

- 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 side 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:

```
var <type> <identifier> [ := <value> ]opt ;
```

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;
#:)

func int my_func<: int x , int y:>(:
    var int res ;
    res := x*y;
    send res;
: )

func driver <: :> (:
    # 'driver' is where the program's execution begins
    # analogous to C's 'main'

    var string s := "my string";
    var bool flag := true;
    flag := false;
```

---

```
var float f := 3.0;
f := to_float(9);
# implicit conversion from int to float is allowed, but it
# is really just syntactic sugar, with the compiler
# still calling the to_float function

var int i;

i := ceil(2.8);
i := floor(3.2);

var int arr[10];
arr[1] := 2;

print<: "Hello World" :>;
for(: int i := 0; i < 10; i = i+1 :)
(:
    print <: "Iteration: ", i :>;
:)
:)
```

---

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");

:)
```