CONSI
LANGUAGE SPECIFICATION DOCUMENT

DOMAIN SPECIFIC LANGUAGE NAME : CONSI
DOMAIN SPECIFIC LANGUAGE ON : CONIC SECTIONS

TEAM MEMBERS :

VIGNAN KOTA – CS21BTECH11029
N SREE HARSHA – CS21BTECH11042
DAVID MALOTH – CS21BTECH11035
BURRA VISHAL MATHEWS – CS21BTECH11010

Indian Institute of Technology
Hyderabad

Contents

1. Introduction

1.1. What is CONSI?. CONSI, also known as "Conic Section DSL," is a language created to simplify the manipulation and analysis of conic sections. These conic sections are shapes like circles, parabolas, ellipses, and hyperbolas widely used in mathematics, physics, engineering, and various scientific fields for practical applications.

The main goal of the CONSI DSL is to make working with these shapes easier by abstracting their complexities and providing an interface. This language offers users data types such as POINT, LINE, CIRCLE, PARABOLA, ELLIPSE, and HYPERBOLA. Additionally, it offers a range of operations and properties that enable users to perform tasks. Some examples include checking if a point lies on a section, obtaining the equation of the section itself, calculating eccentricity values, determining tangent equations at specific points on the conic section curve, or finding normal equations at given points.

1.2. CONSI Uses. Since C/Cpp does not have a dedicated library for conic sections we have created a dsl solely for the usage of conic sections.

- Using a new data type named 'point' to represent points.
- We are trying to implement 4 types of commonly used conic sections and they are, Circles, Parabolas, Ellipses, and Hyperbolas.
- Users can create instances of these conic sections. The CONSI DSL offers a variety of functions that allow users to retrieve and analyze the characteristics of sections.
- These functions include finding equations, calculating eccentricity, determining the equation at a point and finding the normal equation, for a given point, on the conic section.
- CONSI DSL provides advanced analysis tools for conic sections, such as determining the focal length, vertex, and axis of symmetry for ellipses and hyperbolas.
- These features go beyond basic properties and facilitate in-depth exploration of conic sections.

2. Lexical Conventions

2.1. Comments. The comments in this language follow the general C comment syntax.

- With the line ending comments starting with two forward slashes.
  Ex. 2.1: // This is a valid comment
- MultiLine comments are written in the following way "/* Matter */".
  Ex. 2.2: /* This is a valid multiline comment */
- Nested comments and comments between strings are not allowed.
  Ex. 2.3: /* Comments cannot be $*nested*$ like this */

2.2. Whitespaces.

- WhiteSpace characters that are not matched in any regex of the grammar are ignored.
- "\t" is used for a single tab-space, while "\n" is used for new lines in string literals, similar to C.

Both of the programs in Example 2.4 and 2.5 below are equivalent:

```
Ex. 2.4
void main ()
{
   int a = 1;
   float b = 1.1;
   point p = [1,b];
}
```

```
Ex. 2.5
void main()
{int a = 1;
float b = 1.1;
point p = [1,b];}
```

2.3. Reserved Keywords. The following words are reserved keywords in CONSI and cannot be used as identifiers:

if	int	point	NULL	else
float	line	return	break	char
circle	while	continue	string	ellipse
void	for	bool	parabola	hyperbola

2.4. Identifiers. Identifiers in our language need to follow certain rules, and they are:

- The identifier should start with a letter (both capital and small included).
- Then any of the following can be used:
  - Another character.
  - Another underscore.
  - Digit.

The Regular Expression for the identifier is given as follows: `[a-zA-Z]_0-9a-zA-Z]*`.

`foo`, `x_1`, `x1` are valid identifiers.
`1x`, `_x1` are not valid identifiers.

2.5. Punctuators. Punctuation Statements should be terminated with a semicolon (;)

3. Data Types

3.1. Basic Data Types.

Datatype Name	Description	Initialization
int	integer value	`int x = 9;`
float	floating-point number	`float x = 10.1;`
string	sequence of characters	`string s1 = "Hello world";`
char	a character	`char c = '@';` `char d = '1';`
bool	stores either true or false	`bool a = true;` `bool b = false;`

3.2. Conic Related Data Types. Using `[ ]` for declaring elements of a point and `{ }` for declaring elements of line, circle, parabola, hyperbola, ellipse, etc.

Datatype Name	Description	Initialization
point	stores two floating-point numbers	`point p = [1,2];`
line	stores a point & a floating-point number	`line a = {[1,2], 3.1};`
circle	stores a point & a floating-point number	`circle c = {[0,0], 1};`
parabola	stores a point & a line	`parabola x = {[0,0], a};`
ellipse	stores a point & two floating-point numbers	`ellipse y = {[0,0], 1, 2};`
hyperbola	stores a point & two floating-point numbers	`hyperbola z = {[0,0], 2, 3};`

4. Operators and Expressions

Purpose	Symbol	Associativity	Valid Operands
Parentheses for grouping of operations	( )	Left to right	int, float
Member access operators	.	Left to right	circle, parabola, ellipse, hyperbola
Unary negation	NEG	Right to left	all bool
Increment	++	Right to left	int, float
Decrement	–	Right to left	int, float
Exponent	^	Left to right	int, float
Modulo	%	Left to right	int, float
Multiplication	*	Left to right	int, float
Division	/	Left to right	int, float
Addition	+	Left to right	int, float, string
Subtraction	-	Left to right	int, float
Bitwise Operators	|, &	Left to right	-
Intersection	&	Left to right	-
Relational operators	<=, <, >=, >, ==, !=	Left to right	all datatypes
Assignment operators	=, *=, +=, /=, ^, %=	Right to left	wherever the operator is valid
Logical operators	AND, OR	Left to right	bool

5. Declarations

In a program, there are elements like variables, functions, and types. Before using any of these elements, it is necessary to declare them.

5.1. Variable Declaration. A variable declaration starts with the datatype name, followed by the variable name with initialization being optional. Multiple declarations can be separated by commas.

Example 5.1.

```
int a = 5, b;
```

5.2. Function Declaration. A function declaration starts with the keyword `func`, followed by the return type enclosed within parentheses `()`. Then, the function name follows, containing similar syntax rules as an identifier. Inside the parentheses, function parameters with types and names, but without initialization, are specified.

Example 5.2.

```
func (int) add (int a, int b);
```

5.3. Array Declaration. An array declaration is similar to a variable declaration, with differences in size and initialization. The size is specified in square brackets `[]` immediately after the identifier. Initialization can be done by enclosing values in curly braces `{}` separated by commas. The number of values should match the size if initialized.

Example 5.3.

```
int a[10], b[5] = {1, 3, 5, 6, 19};
```

5.4. Declaration Scope. Declaration scope in CONSI is defined by `""`. Any variable, function, or array cannot be used outside of that scope. A variable that already exists in a larger scope cannot be redefined in a smaller scope, meaning they cannot have the same names.

Example 5.4.

```
int a;
int main() {
    int a, b; // Here, 'a' cannot be redeclared in the local scope.
}
```

5.5. Initializations. Every variable, when declared, is initialized to a default value. Variables can be assigned any value within the range of their datatype. The value to be assigned can also be an expression that results in a value.

Example 5.5.

```
int a = 5, b;
```

6. Statements

Types of Statements in CONSI are:

1. Labeled statements
2. Compound statements
3. Expression statements
4. Selection statements
5. Iteration statements

6.1. Labeled Statements. Labeled statements are used in switch statements. A simple identifier followed by a ":" is a label. For example, in switch statements, there are case and default labeled statements.

Example 6.1.

```
case constant-expression : statement
```

6.2. Compound Statements. A compound statement is an optional list of declarations and an optional list of statements enclosed within braces.

Example 6.2.

```
{
    int a; // declaration
    a = 5; // statement
}
```

6.3. Expression Statements. An expression statement is an optional expression followed by a semicolon. If there is some expression, it might have a value. If there is no expression and just a semicolon, it is called an empty or null statement. Statements that have only a function call followed by a semicolon come under this category.

Expressions in CONSI are valid sequences of operands and operators, where operands can be identifiers or constants on which operations are performed.

Example 6.3.

```
a * 5 // Expression using a binary operator, operands are an identifier and a constant
a + b // Expression using a binary operator, both operands are identifiers
a >= 10 // Expression using a relational operator
a /= 10 // Expression using a binary operator
```

6.4. Selection Statements. There are three types of selection statements in CONSI:
1. `if (expression) statement` 2. `if (expression) statement else statement`
3. Switch statements are also a type of selection statements.

Example 6.4.

```
switch (expression) statement
```

6.5. Iteration Statements. There are two types of iteration statements in CONSI:

1. **While Loop**

Example 6.5.1.

```
while (expression)
{
    // statements go here
}
```

2. **For Loop**

Example 6.5.2.

```
for (initialization statement ; condition check expression ; update statement)
{
    // statements go here
}
```

3. **Range-Based For Loops**

Example 6.5.3.

```
for (range declaration : range expression)
{
    // statements go here
}
```

7. Inbuilt Utilities

We plan to implement some standard libraries/headers to provide support for the following domains:

- conic operations