Infix, Prefix and Post fix notations

General format: key is position of operator

		binary	unary
		2 operands and 1 operator	1operand and 1 operator
Infix:	oprand1 operator operand2	x * y	- x
Prefix:	operator oprand1 operand2	* x y	- x
Postfix:	oprand1 operand2 operator	x y *	x -

Infix:	x * y	x * y + z	x - y * z + w / v	(x – y) * z + w / v
Execution	* x y	* x y => r1	/ w v => r1	- x y => r1
Order:		+ r1 z	* y z => r2	(r1) => r2 i.e. r1 == r2
			+ r2 r1 => r3	/ w v => r3
			- x r3	* r2 z => * r1 z => r4
				+ r4 r3
steps:	* x y	+ r1 z	- x r3	+ r4 r3
		+ * x y z	- x + r2 r1	+ * r1 z r3
			- x + * y z r1	+ * r1 z / w v
			- x + * y z / w v	+ * - x y z / w v
Prefix:	* x y	+ * x y z	- x + * y z / w v	+ * - x y z / w v

Compiler statements add x y mul x y

add acc z

Infix, Prefix and Post fix notations

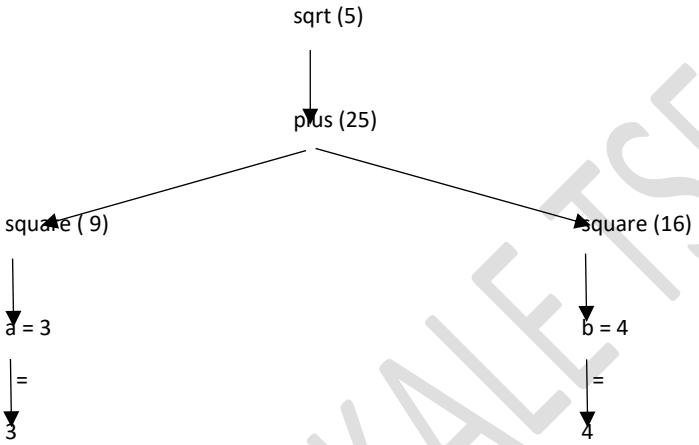
General format: key is position of operator

	2 operands 1 operator			1operand operator	
Infix:	oprand1 operator operand2	a + b	a & b	- a	! a
Prefix:	operator oprand1 operand2	+ a b	& a b	- a	! a
Postfix:	oprand1 operand2 operator	a b +	a b &	a -	a !

Infix:	a + b + c	a + b * c	a – b ^ c * d + e	(a – b) ^ c * d + e
Order:	+ a b => r1	* b c => r1	^ b c => r1	- a b => r1
	+ r1 c	+ a r1	* r1 d => r2	(r1) => r2 i.e r1 => r2 No brackets
			+ r2 e => r3	^ r2 c => r3
			- a r3	* r3 d => r4
				+ r4 e
Prefix:	+ r1 c	+ a r1	- a r3	+ r4 e
steps:	+ + a b c	+ a * b c	- a + r2 e	+ * r3 d e
			- a + * r1 d e	+ * ^ r2 c d e
			- a + * ^ b c d e	+ * ^ r1 c d e
				+ * ^ - a b c d e
Prefix:	+ + a b c	+ a * b c	- a + * ^ b c d e	+ * ^ - a b c d e

Lambda Function Evaluation : Let is used to assign values or function

Function	Remark
(let ((a 3)	a = 3
(b 4)	b = 4
(square (lambda (x) (* x x)))	square is the name of lambda expression x*x over input parameter x.
(plus +))	plus is the name od addition operator
(sqrt (plus (square a) (square b))))	sqrt is a standard function



Scope :

```
(let ((a 3))    a = 3
(let ((a 4)     a = 4  ;   b = a  i.e b = 4;
(b a))          "comment def of a = 4 ends"
(+ a b)) ==> 7   "for evaluation we have a = 3 and b = 4"
```

Recursion:

```
(letrec ((fib
(lambda (n)
(if (= n 0) 1
 if (= n 1) 1
 (+  ( fib (- n 1) )  (fib (- n 2) )  )
))))
```