**PUNTO 1. INFIJO -> POSTFIJO**

**A.**

(X + Y) \* (Z ^ (S + T) + U) - V

(X Y +) \* (Z ^ (S T +) + U) - V

(X Y +) \* (Z S T + ^) + U) - V

((X Y +) \* (Z S T + ^ U +)) - V

X Y + Z S T + ^ U + \* - V

X Y + Z S T + ^ U + \* V -

**R/ X Y + Z S T + ^ U + \* V -**

**B.**

A + ((X + Y) \* (W + V)) ^ (I + J)

A + ((X Y +) \* (W V +)) ^ (I + J)

A + ((X Y + W V + \*) ^ (I + J))

A + (X Y + W V + \* I J + ^)

A X Y + W V + \* I J + ^ +

**R/A X Y + W V + \* I J + ^ +**

**C.**

(A + B) \* (C + D)

(A B +) \* (C D +)

**A B + C D \***

**R/A B + C D \***

**D.**

((A + B) - C)

((A B +) - C)

A B + C -

**R/A B + C -**

**E.**

(A + B) \* (C - D) \* E \* F

((A B +) \* (C D -) \* E \* F)

(((A B + C D - \*) \* E) \* F)

((A B + C D - \* E \*) \* F)

A B + C D - \* E \* F \*

**R/A B + C D - \* E \* F \***

**F.**

A + B) \* (C / (D - E) + F) - G

(A B +) \* ((C / (D E -)) + F) - G

(A B +) \* ((C D E - /) + F) - G

((A B +) \* (C D E - / F +)) - G

((A B + C D E - F + \*) - G)

A B + C D E - / F + \* G -

**R/A B + C D E - / F + \* G -**

**G.**

((A ^ B) ^ C)

(A B ^) ^ C

A B ^ C ^

**R/A B ^ C ^**

**H.**

A ^ B \* C - D + E / F / (G + H)

((((A ^ B) \* C) - D) + ((E / F) / (G H +)))

((((A B ^) \* C) - D) + ((E F /) / (G H +)))

((((A B ^ C \*) - D) + ((E F /) / (G H +)))

(((A B ^ C \*) - D) + (E F / G H + /))

((A B ^ C \* D -) + (E F / G H + /))

A B ^ C \* D - E F / G H + / +

**R/A B ^ C \* D - E F / G H + / +**

**I.**

A + (((B - C) \* (D - E) + F) / G) ^ (H - J)

A + (((B C -) \* (D E -) + F) / G) ^ (H J -)

A + ((B C - D E - \* F) / G) ^ (H J -)

A + ((B C - D E - \* F G /) ^ (H J -))

(A + (B C - D E - \* F G /)) ^ (H J -)

((A B C - D E - \* F G / +)  ^ (H J -))

A B C - D E - \* F G / + H J - ^

**R/A B C - D E - \* F G / + H J - ^**

**PUNTO 2. INFIJO -> PREFIJO**

**A.**

(X + Y) \* (Z ^ (S + T) + U) - V

(+ X Y) \* ((Z ^ (+ S T)) + U) - V

(+ X Y) \* ((^ Z + S T)) + U) - V

(+ X Y) \* ((+ ^ Z + S T U) - V)

((+ X Y) \* (- + ^ Z + S T U V))

\* + X Y - + ^ Z + S T U V

**R/\* + X Y - + ^ Z + S T U V**

**B.**

A + ((X + Y) \* (W + V)) ^ (I + J)

A + ((+ X Y) \* (+ W V)) ^ (+ I J)

(A + (\* + X Y + W V)) ^ (+ I J)

((+ A \* + X Y + W V) ^ (+ I J))

^ + A \* + X Y + W V + I J

**R/^ + A \* + X Y + W V + I J**

**C.**

(A + B) \* (C + D)

(+ A B) \* (+ C D)

\* + A B + C D

**R/\* + A B + C D**

**D.**

((A + B) - C)

((+ A B) - C)

- + A B C

**R/- + A B C**

**E.**

(A + B) \* (C - D) \* E \* F

((+ A B) \* (- C D)) \* E \* F

((\* + A B - C D) \* E) \* F

(\* \* + A B - C D E) \* F

\* \* \* + A B - C D E F

**R/\* \* \* + A B - C D E F**

**F.**

(A + B) \* (C / (D - E) + F) - G

(+ A B) \* ((C / (- D E)) + F) - G

(+ A B) \* ((/ C - D E) + F) - G

((+ A B) \* (+ / C - D E F)) - G

((\* + A B + / C - D E F) - G)

- \* + A B + / C - D E F G

**R/- \* + A B + / C - D E F G**

**G.**

((A ^ B) ^ C)

((^ A B) ^ C)

^ ^ A B C

**R/^ ^ A B C**

**H.**

A ^ B \* C - D + E / F / (G + H)

(((^ A B) \* C) - D) + (E / (F / (G + H)))

((\* ^ A B C) - D) + (E / (F / (G + H)))

(- \* ^ A B C D) + (E / (F / (G + H)))

(- \* ^ A B C D) + (E / (F / (+ G H)))

(- \* ^ A B C D) + (E / (/ F + G H))

(- \* ^ A B C D) + (/ / E F + G H)

+ - \* ^ A B C D // E F + G H

**R/+ - \* ^ A B C D // E F + G H**

**I.**

A + (((B - C) \* (D - E) + F) / G) ^ (H - J)

A + ((((- B C) \* (- D E)) + F) / G) ^ (- H J)

A + (((\* - B C - D E) + F) / G) ^ (- H J)

A + ((+ \* - B C - D E F) / G) ^ (- H J)

(A + (/ + \* - B C - D E F G)) ^ (- H J)

(+ A / + \* - B C - D E F G) ^ (- H J)

^ + A / + \* - B C - D E F G - H J

**R/^ + A / + \* - B C - D E F G - H J**

**PUNTO 3.  PREFIJO -> INFIJO**

**A.**

+ - ^ X Y Z \* I \* \* K L M

((X ^ Y) - Z) + (I \* ((K \* L) \* M))

**R/((X ^ Y) - Z) + (I \* ((K \* L) \* M))**

**B.**

+ + L - \* ^ M N O / + P Q \* R S T

(L + ((M ^ N) \* O)) + ((P + Q) - (R \* S) / T)

**R/(L + ((M ^ N) \* O)) + ((P + Q) - (R \* S) / T)**

**PUNTO 4. POSFIJO -> INFIJO**

**A.**

P Q R S T - + ^ U V \* -

(P \* (Q ^ (R +(S + T)))) - (U \* V)

**R/(P \* (Q ^ (R +(S + T)))) - (U \* V)**

**B.**

L M - N + O P Q - + ^

((L - M) + N) ^ (O + (P - Q))

**R/((L - M) + N) ^ (O + (P - Q))**

**INVESTIGACIÓN**

**1. QUÉ ES PROGRAMACIÓN FUNCIONAL Y PARA QUÉ SIRVE?**

Es un paradigma de programación declarativa basado en el uso de funciones matemáticas, en contraste con la programación imperativa, que enfatiza los cambios de estado mediante la mutación de variables. La programación funcional tiene sus raíces en el cálculo lambda, un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión. Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.

Los lenguajes de programación funcional, especialmente los puramente funcionales, han sido enfatizados en el ambiente académico y no tanto en el desarrollo comercial o industrial. Sin embargo, lenguajes de programación funcional como Scheme, Erlang, Rust, Objective Caml , Scala, F# y Haskell, han sido utilizados en aplicaciones comerciales e industriales por muchas organizaciones. La programación funcional también es utilizada en la industria a través de lenguajes de dominio específico como R (estadística), Mathematica (matemáticas simbólicas), J y K (análisis financiero).

Los lenguajes de uso específico usados comúnmente como SQL y Lex/Yacc, utilizan algunos elementos de programación funcional, especialmente al procesar valores mutables. Las hojas de cálculo también pueden ser consideradas lenguajes de programación funcional.

La programación funcional también puede ser desarrollada en lenguajes que no están diseñados específicamente para la programación funcional. En el caso de Perl, por ejemplo, que es un lenguaje de programación imperativo, existe un libro que describe como aplicar conceptos de programación funcional. JavaScript, uno de los lenguajes más ampliamente utilizados en la actualidad, también incorpora capacidades de programación funcional. Python también incorpora particularidades de los lenguajes funcionales como listas de comprensión y funciones de tratamiento de listas como matemática de conjuntos. Java en su versión 8, está incorporando la programación funcional, así como el uso de las expresiones lambda.

**2. QUE ES RACKET?**

Es un lenguaje de programación de amplio espectro de la familia de Lisp y Scheme. Es multiparadigma así como de propósito general. Uno de sus principales objetivos tras su diseño es posibilitar la creación de nuevos lenguajes o dialectos.[1](https://es.wikipedia.org/wiki/Racket_(lenguaje_de_programaci%C3%B3n)#cite_note-1) El lenguaje es usado en una variedad de entornos tales como scripting, enseñanza en ingeniería informática o la investigación.

La plataforma nos ofrece la herramienta DrRacket, un entorno de desarrollo integrado programado en Racket, que nos facilitará la tarea de programar en Racket. También nos ofrece raco, un herramienta para la línea de comandos que nos permitirá instalar paquetes o compilar librerías.[2](https://es.wikipedia.org/wiki/Racket_(lenguaje_de_programaci%C3%B3n)#cite_note-Welcome_to_Racket-2)

La plataforma se adhiere a la iniciativa Software libre, y está liberada bajo una licencia LGPL.

Racket es altamente flexible, incluso sin el uso de dialectos. Características tales como el uso de macros, módulos, clausuras, recursión en cola, y mucho más, le permiten ser usado para la realización de todo tipo de tareas, desde generación de gráficos a web scrapers. Además, el potente sistema de macros permite a los desarrolladores controlar todos los aspectos de un lenguaje, una de las principales metas tras su diseño. El framework viene con un gestor de paquetes llamado PLaneT integrado con el sistema de módulos, lo que permite a los usuarios importar y usar librerías de terceros de forma totalmente transparente.

La principal característica de Racket es la capacidad de crear y extender nuevos lenguajes. Esto es posible gracias a una serie de componentes:

* Un sistema de módulos capaz de enlazar código y manejar [e](https://es.wikipedia.org/wiki/Espacio_de_nombres)spacios de nombres
* Un sistema de macros que permite la creación de nuevas formas sintácticas
* Un sistema de tiempo de ejecución que ofrece muchas opciones a los nuevos lenguajes
* Formas de especificar e implementar analizadores sintácticos para dichos lenguajes.

**3. CÓMO SE IMPLEMENTA Y FUNCIONA UNA PILA?**

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. Esta estructura se aplica en multitud de ocasiones en el área de informática debido a su simplicidad y ordenación implícita de la propia estructura.

Para el manejo de los datos se cuenta con dos operaciones básicas: apilar (push), que coloca un objeto en la pila, y su operación inversa, retirar (o desapilar, pop), que retira el último elemento apilado.

Por analogía con objetos cotidianos, una operación apilar equivaldría a colocar un plato sobre una pila de platos, y una operación retirar a retirarlo.

[LINK IMPLEMENTACION DE LA PILA](http://blog.martincruz.me/2012/10/pilas-en-c.html)

**4.** **ALGORITMO PARA PASAR DE POSTFIJO A PREFIJO Y DE PREFIJO A POSTFIJO SIN IR A INFIJO.**

[LINK POSTFIJO->PREFIJO](https://pastebin.com/UubqsAYy)

[LINK PREFIJO->POSTFIJIO](https://pastebin.com/C90bejp5)