*****Instituto Politécnico Nacional***

***Escuela Superior de Cómputo***

*Aplicaciones para Comunicaciones de Red*

***Practica 7: Calculadora Distribuida***

***Integrantes:***

* *Ornelas García Luis Ángel*
* *Sampayo Hernández Mauro*

***Grupo:*** *3CM5*

***Profesor:*** *Moreno Cervantes Axel Ernesto*

**Introducción:**

Los sockets son una excelente opción para el desarrollo de aplicaciones en red, ya que son flexibles y pueden adaptarse bien a prácticamente cualquier uso que se les desee dar. Sin embargo, no son la única forma de desarrollar aplicaciones cliente-servidor o distribuidas. Existen interfaces de programación de aplicaciones de más alto nivel que ofrecen un mayor nivel de abstracción y simplicidad para el programador, además de transparencia para comunicar aplicaciones. Para el desarrollo de esta práctica utilizaremos XDR (External Data Representation), el cual es un estándar (rfc 1832) para la descripción y codificación de datos. Es muy útil para transferir datos entre sistemas con diferentes arquitecturas, tales como estaciones de trabajo SUN, VAX, Cray, etc. XDR usa un lenguaje para describir formatos de datos. No es un lenguaje de programación. Su sintaxis es similar a lenguaje C.

**Desarrollo:**

Para la realización de esta práctica, se crearon 2 ficheros IDL con extensión .x llamados **“Arithmetic.x”** y **“Trigonometric.x”**.

En **“Arithmetic.x”** se definió una estructura llamada “operandos” la cual contiene dos variables flotantes llamadas “operando1” y “operando2”, y se especificó el nombre del servicio (ARITHMETIC) y su número de versión. Los servicios declarados para este servidor son operaciones aritméticas (suma, resta, multiplicación, división y exponenciación) los cuales recibirán datos para la estructura “operandos”.

En **“Trigonometric.x”** se especificó el nombre del servicio (TRIGONOMETRIC) y su número de versión. Los servicios declarados para este servidor son operaciones trigonométricas y funciones matemáticas que solo reciben un operando flotante (seno, coseno, tangente, secante, cosecante, cotangente, logaritmo y raíz cuadrada).

Se procede a generar las stubs y las plantillas mediante rpcgen de **“Arithmetic.x”** y **“Trigonometric.x”**; así, se generan los siguientes archivos: **“Trigonometric\_client.c”**, **“Arithmetic\_xdr.c”**, **“Arithmetic.h”**, **“Arithmetic\_server.c”**, **“Arithmetic\_svc.c”**, **“Arithmetic\_client.c”**, **“Arithmetic\_clnt.c”**,para **“Arithmetic.x”**; y **“Makefile.Trigonometric”**, **“Trigonometric.h”**, **“Trigonometric\_server.c”**, **“Trigonometric\_svc.c”**, **“Trigonometric\_client.c”**, **“Trigonometric\_clnt.c”**,para **“Arithmetic.x”**.

Los archivos **“Arithmetic\_client.c”** y **“Trigonometric\_client.c”** fueron sustituidos para que sus respectivos servidores actúen sobre un mismo cliente, el cual se llama **“Calculadora.c”**; también los archivos **“Trigonometric\_client.c”** y **“Makefile.Trigonometric”** fueron modificados para adaptarse a lo anteriormente mencionado, así como también incluir la librería “Math.h” y la estructura de datos Pila de memoria dinámica y cuyas funciones están implementadas en el archivo **“TADPilaDin.c”** y la librería **“TADPilaDin.h”.**

Las funciones contenidas en los archivos **“TADPilaDin.c”** y **“TADPilaDin.h”** se enlistan a continuación:

* **void Initialize(pila \*s):** Inicializa una pila para su uso.
* **void Push(pila \*s, elemento e):** Introduce un elemento a la pila.
* **void Pop(pila \*s):** Extrae un elemento de la pila.
* **boolean Empty(pila \*s):** Devuelve ‘1’ si la pila está vacía, si no devuelve ‘0’.
* **elemento Top(pila \*s):** Obtiene el "elemento" del tope de la pila sin extraerlo de esta.
* **int Size(pila \*s):** Devuelve el número de elementos de la pila.
* **void Destroy(pila \*s):** Borra a todos los elementos en a la pila de memoria.

En **“TADPilaDin.h”** se encuentra la estructura “elemento” la cual tiene una variable tipo entero, una de tipo flotante, una de tipo carácter y un string de longitud 4. Estas variables fueron declaradas dentro de dicha estructura pues serán utilizadas en el programa cliente **“Calculadora.c”**.

Finalmente, en los archivos **“Arithmetic\_server.c”** y **“Trigonometric\_server.c”** se programaron cada una de las funciones que llevarán a cabo las operaciones matemáticas de la Calculadora.

A continuación, se describen las funciones contenidas en el programa cliente “**Calculadora.c”.**

* **Calculadora.c:**

Este programa actúa como el cliente, y es el que llevará a cabo toda la lógica tras el funcionamiento de la calculadora y la evaluación de las expresiones infijas, postfijas y prefijas.

**Función main():**

Al momento de ejecutar el programa se le debe de indicar al programa a que dirección de red desea conectarse como segundo argumento de la ejecución.

Al inicializarse la ejecución del programa se verificará que el argumento de la dirección haya sido ingresado y este será guardado en un arreglo dinámico de caracteres global llamada “host”.

Después, el programa entrará a un ciclo infinito, el cuál ejecutará la función **“Menu()”** para desplegar el menú de la calculadora al usuario, y se le pedirá que seleccione una de 4 opciones. Las opciones a elegir son:

1. Evaluar expresión prefija
2. Evaluar expresión postfija
3. Evaluar expresión infija
4. Salir

Si el usuario selecciona la opción número 4, el programa finalizará su ejecución sin errores de ningún tipo. De lo contrario si se selecciona alguna de las otras opciones, al usuario se le pedirá que introduzca una expresión válida que corresponda al tipo de expresión que haya decidido evaluar.

Si el usuario eligió la primera opción se ejecutará la función **“Prefijo()”**, si eligió la segunda opción se ejecutará la función **“Postfijo()”**; en ambos casos a la función que sea ejecutada se le enviará la cadena de caracteres de la expresión a evaluar.

Por último, si el usuario selecciono tercera opción, se ejecutará la función **“ConvertirInfijoPostfijo()”** enviándole la cadena de caracteres de la expresión a evaluar, y posteriormente se ejecutará la función **“Postfijo()”** a la cual se le enviará la cadena que haya sido devuelta por la función **“ConvertirInfijoPostfijo()”**.Esto se realiza debido a que no hay una manera viable de computar la evaluación de una expresión infija directamente

Si se introduce una opción inválida, el programa mandará un mensaje de error y se ejecutará la siguiente iteración del ciclo.

**Función Menú():**

Esta función se encarga de mostrar el menú de la Calculadora, el cuál consta de una tabla que muestra la lista de operaciones que se pueden introducir y las opciones que el usuario puede elegir para llevar a cabo la evaluación de la expresión que vaya a ser introducida en la calculadora.

**Función Postfijo():**

Esta función lleva a cabo la evaluación de expresiones postfijas. Al ejecutarse se inicializa una pila dinámica y 3 estructuras de tipo “elemento”. Tras esto se ejecuta un ciclo que tendrá tantas iteraciones como caracteres tenga la expresión a evaluar.

Dentro del ciclo se realiza el análisis de cada carácter de la expresión para determinar qué tipo de carácter son.

Si el carácter es un número se inicializará una cadena de caracteres de longitud 50 y se iniciará un ciclo en el cual se guardará dentro de la cadena de caracteres tanto el carácter analizado, como todos los caracteres que se encuentren después de este siempre y cuando estos sean también números o un punto(.) para el caso de que el número en cuestión sea de tipo flotante. Seguidamente se realiza la conversión del arreglo de caracteres a un número flotante y se guarda en un elemento para posteriormente insertar dicho elemento en la pila y ajustar el ciclo para que continúe el análisis de los caracteres desde el carácter siguiente al último carácter guardado en el ciclo.

Si el carácter es una letra minúscula se inicializará una cadena de caracteres de longitud 4 y se iniciará un ciclo en el cual se guardará dentro de la cadena de caracteres tanto el carácter analizado, como todos los caracteres que se encuentren después de este siempre y cuando estos sean también letras minúsculas y la cadena de caracteres aún no este completamente llena. Seguidamente se extrae un elemento de la pila y se guarda en un elemento auxiliar, para después mandarlo a la función **“FunctTrigonometricas()”** junto a la cadena de caracteres para realizar la función matemática correspondiente. El resultado que devuelva esta función será guardado en un elemento para después insertarlo en la pila y ajustar el ciclo para que continúe el análisis de los caracteres desde el carácter siguiente al último carácter guardado en el ciclo.

Si el carácter es un signo de puntuación, se comprobará si este es una coma (,) y en caso de serlo, se pasará a la siguiente iteración del ciclo directamente. En caso contrario, se extraerán dos elementos de la pila que serán guardados en dos estructuras elemento auxiliares, para después ser mandadas junto a el carácter evaluado a la función **“FunctAritmeticas()”** para realizar la operación matemática correspondiente. El resultado que devuelva esta función será guardado en un elemento e insertado en la pila.

Si el carácter no cumple con ninguno de los 3 requisitos mencionados anteriormente, el programa devolverá un mensaje de error y se detendrá la ejecución del programa.

Una vez finalice el ciclo, se extraerá el único elemento que quede dentro de la pila (el cuál es el resultado de la expresión matemática evaluada en el ciclo) y se imprimirá en pantalla el valor flotante que contenga la estructura.

**Función Prefijo():**

Esta función lleva a cabo la evaluación de expresiones prefijas y realizará el mismo procedimiento que la función **“Postfijo()”** con la diferencia de que el ciclo irá analizando los caracteres de la expresión en orden inverso.

Adicionalmente, al estarse analizando los caracteres de manera inversa, se necesitará llevar a cabo la inversión de las cadenas de caracteres que son inicializadas si el carácter es un número o una letra minúscula para así poder obtener el valor o palabra correcta al realizar su análisis. Dicho procedimiento se lleva a cabo en la función **“InvertirCadena()”**.

**Función ConvertirInfijoPostfijo():**

Esta función realiza una conversión de notación infija a postfija. Al ejecutarse se inicializa una pila dinámica, 3 estructuras de tipo “elemento”, dos enteros que ayudarán a determinar la prioridad de los caracteres que representen una operación o función matemática y una nueva cadena de caracteres para guardar la expresión ya convertida en postfijo llamada “postfijo”.

Cabe resaltar que la prioridad que tendrá cada operación matemática será la siguiente:

* **Funciones trigonométricas y matemáticas**: Prioridad 3
* **Exponenciación (^):** Prioridad 2
* **Multiplicación (\*) y División (/):** Prioridad 1
* **Suma (+) y Resta (-):** Prioridad 0

Se manda la expresión a evaluar a la función **“Parentesis()”** para comprobar que la notación de los paréntesis de la expresión infija sean correctos. En caso de no ser correctos, el programa devolverá un mensaje de error y finalizará su ejecución.

Tras esto se ejecuta un ciclo que tendrá tantas iteraciones como caracteres tenga la expresión a evaluar. Dentro del ciclo se realiza el análisis de cada carácter de la expresión para determinar qué tipo de carácter son.

Si el carácter es un número se iniciará un ciclo en el cual se guardará dentro de la cadena de caracteres “postfijo” tanto el carácter analizado, como todos los caracteres que se encuentren después de este siempre y cuando estos sean también números o un punto(.) para el caso de que el número en cuestión sea de tipo flotante. Finalmente, una vez el ciclo termine se añadirá una coma (,) después del número evaluado y se ajustará el ciclo para que continúe el análisis de los caracteres desde el carácter siguiente al último carácter guardado en el ciclo.

Si el carácter es una letra minúscula se inicializará una cadena de caracteres de longitud 4 y se iniciará un ciclo en el cual se guardará dentro de la cadena de caracteres tanto el carácter analizado, como todos los caracteres que se encuentren después de este siempre y cuando estos sean también letras minúsculas y la cadena de caracteres aún no esté completamente llena. Seguidamente se asigna el valor 5 a la variable entera que representa la prioridad, se copia la cadena de caracteres a un elemento y se inserta dicho elemento a la pila. Finalmente, se ajusta el ciclo para que continúe el análisis de los caracteres desde el carácter siguiente al último carácter guardado en el ciclo.

Si el carácter es un signo de puntuación, esté será copiado dentro de un elemento y dependiendo que signo de puntuación sea se realizará determinada acción:

* Si el carácter es un paréntesis de abertura “(“, el elemento que lo contenga será insertado en la pila y se reiniciará la prioridad (el valor entero de la prioridad será asignado como ‘0’).
* Si el carácter es un paréntesis de cerradura “)”, se iniciará un ciclo en el cual se extraerán todos los elementos de la pila hasta encontrar un paréntesis de abertura “(“, comprobando si el elemento extraído contiene un carácter (almacena una operación aritmética) o una cadena de caracteres (almacena una función matemática), para después añadir dicho carácter o dichos caracteres en “postfijo” seguidos de una coma. Una vez finalizado el ciclo se extraerá el paréntesis de abertura y si la pila no esta vacía aún, se determinará la prioridad del símbolo que se encuentre en el tope de la pila (debido a que la prioridad fue reseteada cuando entro el paréntesis de entrada).
* Si el carácter no es un paréntesis, se analiza si éste se trata de un operador aritmético (+, -, \*, / o ^) y de serlo, se le asignará el valor correspondiente a un entero de prioridad auxiliar. De no tratarse el carácter de un operador matemático, el programa mostrará un mensaje de error y finalizará su ejecución.

Posterior a esto se comprobará si la prioridad es menor o igual a el valor de la prioridad auxiliar, y en caso de serlo, a la prioridad se le asignará el valor de la prioridad auxiliar, e insertará el elemento que contiene el operador matemático a la pila. En caso contrario se iniciará un ciclo en el cual se extraerán todos los elementos de la pila hasta que la pila quede completamente vacía o hasta encontrar un paréntesis de abertura “(“; comprobando si el elemento extraído contiene un carácter (almacena una operación aritmética) o una cadena de caracteres (almacena una función matemática), para después añadir dicho carácter o dichos caracteres en “postfijo” seguidos de una coma. Una vez finalizado el ciclo se insertará el elemento que contiene el operador matemático a la pila y se determinará la prioridad de dicho operador.

Si el carácter no cumple con ninguno de los 3 requisitos mencionados anteriormente, el programa devolverá un mensaje de error y se detendrá la ejecución del programa.

Una vez finalice el ciclo, se comprobará si la pila está vacía, y en caso de no estarlo, se iniciará un ciclo en el cual se extraerán todos los elementos de la pila hasta que la pila quede completamente vacía, comprobando si el elemento extraído contiene un carácter (almacena una operación aritmética) o una cadena de caracteres (almacena una función matemática), para después añadir dicho carácter o dichos caracteres en “postfijo” seguidos de una coma.

Finalmente se retorna el resultado de la conversión de la expresión a postfijo.

**Función Paréntesis():**

Esta función comprueba si el orden de los paréntesis de una expresión infija es correcto o no. Al ejecutarse esta función se inicializa una pila y un elemento; y se iniciará un ciclo que tendrá tantas iteraciones como caracteres tenga la expresión infija y en el cual se analizarán cada uno de los caracteres de dicha expresión.

Si el carácter es un paréntesis de apertura “(“ este será metido a un elemento que posteriormente será insertado a la pila. Si el carácter es un paréntesis de cerradura “)“ se extraerá un elemento de la pila. S el carácter no es un paréntesis no se realizará ninguna acción.

Si al terminar el ciclo, la pila se encuentra completamente vacía, se destruye la pila y se devuelve el valor ’1’. En caso contrario, si la pila sigue llena para el final del ciclo o si al momento de recibir un paréntesis de cerradura durante el ciclo la pila se encuentra vacía, se devolverá el valor ‘0’.

**Función InvertirCadena():**

Esta función se encarga de invertir una cadena de caracteres por medio de un ciclo que inicia con una variable i en la posición 0 de la cadena y una variable j en la posición del ultimo carácter de la cadena. En cada iteración se guarda el carácter de la posición i de la cadena en una variable temporal, luego el carácter en dicha posición será sustituido por el carácter que se encuentre en la posición j de la cadena y después el carácter de esa posición será sustituido por el de la variable temporal. Finalmente se aumenta la variable i y se decrementa la variable j. Este ciclo continuara mientras i sea menos que la longitud de la cadena dividida entre 2.

**Función FunctTrigonometricas():**

Esta función se encarga de gestionar todas las funciones trigonométricas y matemáticas, así como también establecer conexión con el servidor para su posterior cálculo. Al ejecutarse esta función se inicializa una variable de tipo CLIENT\*, una variable flotante para guardar el resultado de la operación y otra variable flotante la cual será igualada a la variable flotante contenida en la estructura elemento recibida por la función.

Se realiza la llamada a la función ***“clnt\_create()”*** la cual realiza el proceso de enlace con el servidor (en este caso el servidor para las funciones trigonométricas). En caso de que se produzca un error en el enlace con el servidor, se llamara a la función   
***“clnt\_pcreateerror()”*** la cuál imprimirá la información sobre el error y finalizará la ejecución del programa.

Se realiza una comparación de la cadena de texto recibida por la función para corroborar que se trata de una función válida y determinar cuál función es, para posteriormente realizar el llamado a la función correspondiente del servidor pasándole como parámetro el valor de la variable flotante, para después guardar el resultado retornado por esta. En caso de que la cadena de texto no coincida con ninguna función válida, se mostrará un mensaje de error y el programa finalizará su ejecución.

Finalmente se ejecuta la función ***“clnt\_destroy()”*** la cual rompe el enlace con el servidor y se retorna el valor del resultado de la operación.

**Función FunctAritmeticas():**

Esta función se encarga de gestionar todas las operaciones aritméticas, así como también establecer conexión con el servidor para su posterior cálculo. Al ejecutarse esta función se inicializa una variable de tipo CLIENT\*, una variable flotante para guardar el resultado de la operación y una estructura operandos en donde se guardarán los valores flotantes de los elementos recibidos por la función, en sus respectivas variables flotantes operando1 y operando2.

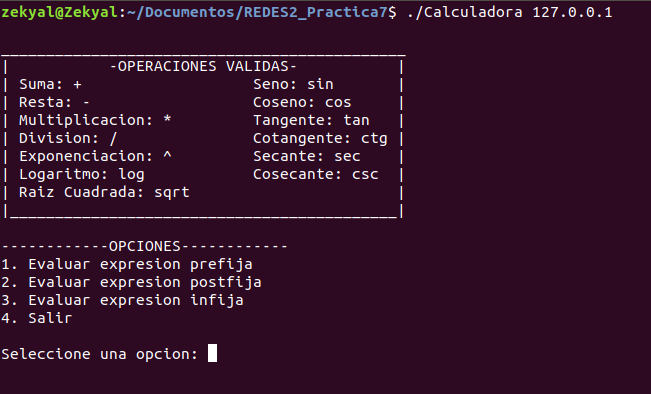
Se realiza la llamada a la función ***“clnt\_create()”*** la cual realiza el proceso de enlace con el servidor (en este caso el servidor para las funciones aritméticas). En caso de que se produzca un error en el enlace con el servidor, se llamara a la función   
***“clnt\_pcreateerror()”*** la cuál imprimirá la información sobre el error y finalizará la ejecución del programa.

Se realiza una comparación del carácter recibido por la función para corroborar que este represente el símbolo de una operación aritmética válida, para posteriormente realizar el llamado a la función correspondiente del servidor pasándole como parámetro la estructura en donde fueron guardadas las dos variables flotantes, para después guardar el resultado retornado por esta. En caso de que el carácter no coincida con ningún símbolo válido, se mostrará un mensaje de error y el programa finalizará su ejecución.

Finalmente se ejecuta la función ***“clnt\_destroy()”*** la cual rompe el enlace con el servidor y se retorna el valor del resultado de la operación.

**Pruebas:**

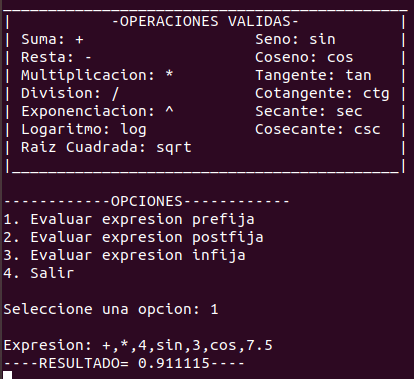
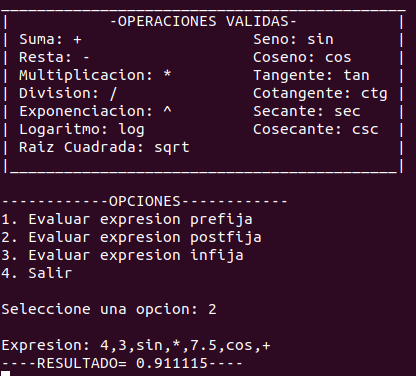
Al ejecutarse la calculadora, se muestra este menú:

****

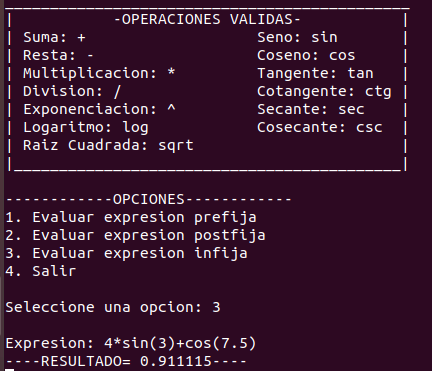
A continuación, se enlistan 4 funciones escritas cada una en expresión infija, postfija y prefija e insertadas en la calculadora, para demostrar la funcionalidad de cada una de las operaciones válidas. Por conveniencia las funciones se enlistarán en infijo:

* )

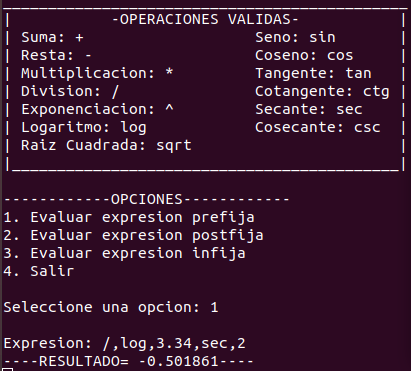
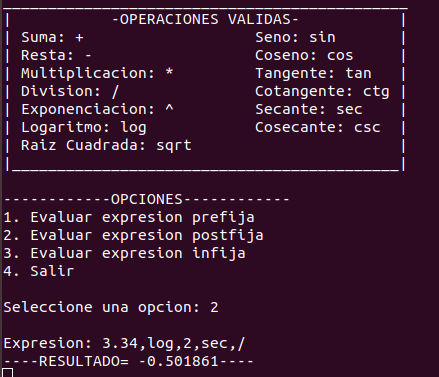
**Expresión Prefija: Expresión Postfija:**

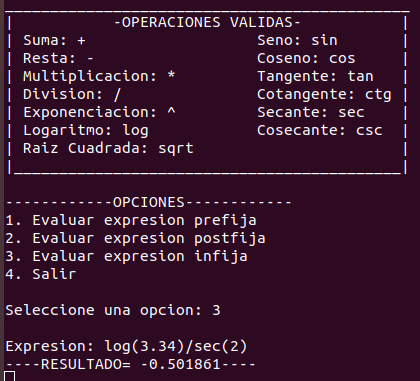
**Expresión Infija:**



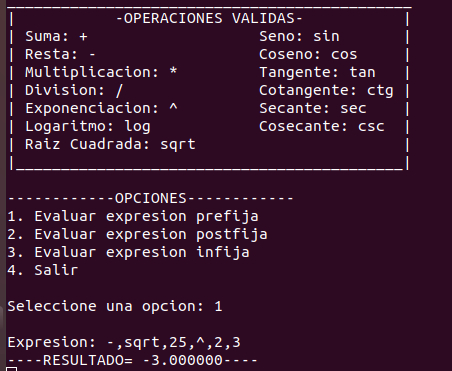
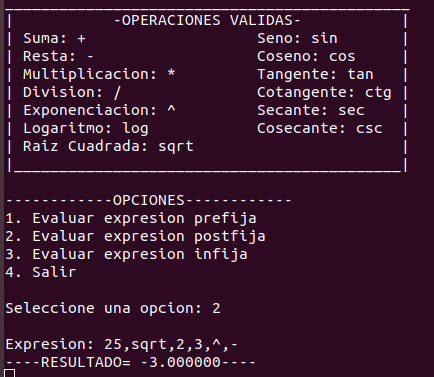
**Expresión Prefija: Expresión Postfija:**

** **

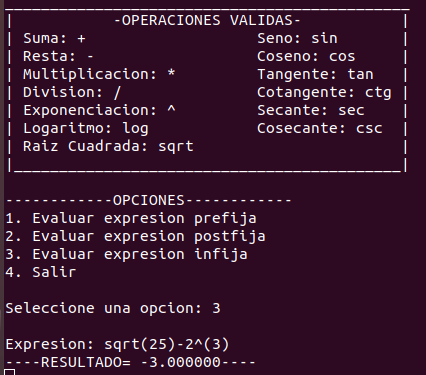
**Expresión Infija:**

****

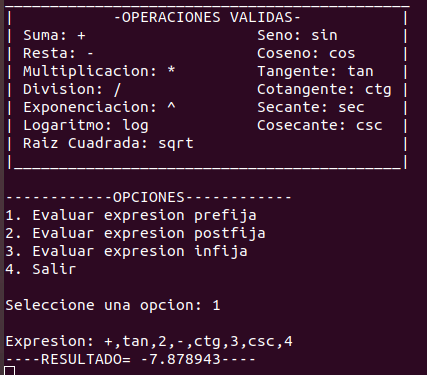
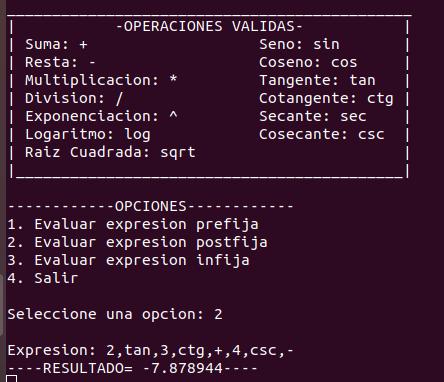
**Expresión Prefija: Expresión Postfija:**

** **

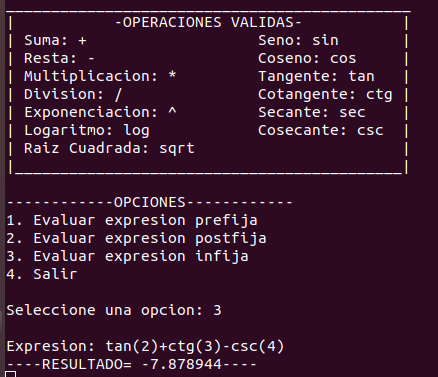
**Expresión Infija:**

****

**Expresión Prefija: Expresión Postfija:**

** **

**Expresión Infija:**

****

**Conclusión:**

El protocolo XDR es una herramienta bastante útil para la creación de librerías de funciones y la transferencia de datos entre máquinas independientemente de el sistema operativo que estas utilicen o la arquitectura que estas posean. Esto es de gran ayuda, ya que le brinda una mayor portabilidad a las aplicaciones que implementen este protocolo.

**Bibliografía:**

* <http://laurel.datsi.fi.upm.es/~ssoo/SOD.dir/practicas/guiarpc.html#caso5>
* <http://www.neptalium.com/blog/tag/notacion-prefija/>
* <https://runestone.academy/runestone/static/pythoned/BasicDS/ExpresionesInfijasPrefijasYSufijas.html>