INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

ESTRUCTURAS DE DATOS

PROF. FRANCO MARTÍNEZ EDGARDO ADRIÁN

EVALUACIÓN DE EC. INFIJA

PRACTICA 1 – TAD PILA

1CM12

COLABORADORES: (VAPORUB.EXE)

AZPEITIA HERNÁNDEZ VLADIMIR

GÓMEZ RODRÍGUEZ EDUARDO

RAYA TOLENTINO PAOLA

Introducción

La práctica 1 “TAD Pila”, busca que se pueda implementar una pila dinámica y estática desarrollada en lenguaje C, con el objetivo de que se use en algún problema de utilidad, de manera que una pila sea la solución a dicha problemática, de manera que el programa buscado es uno en el que el usuario introduzca una expresión matemática infija con paréntesis y jerarquías, y el programa arroje el resultado correcto de la expresión.

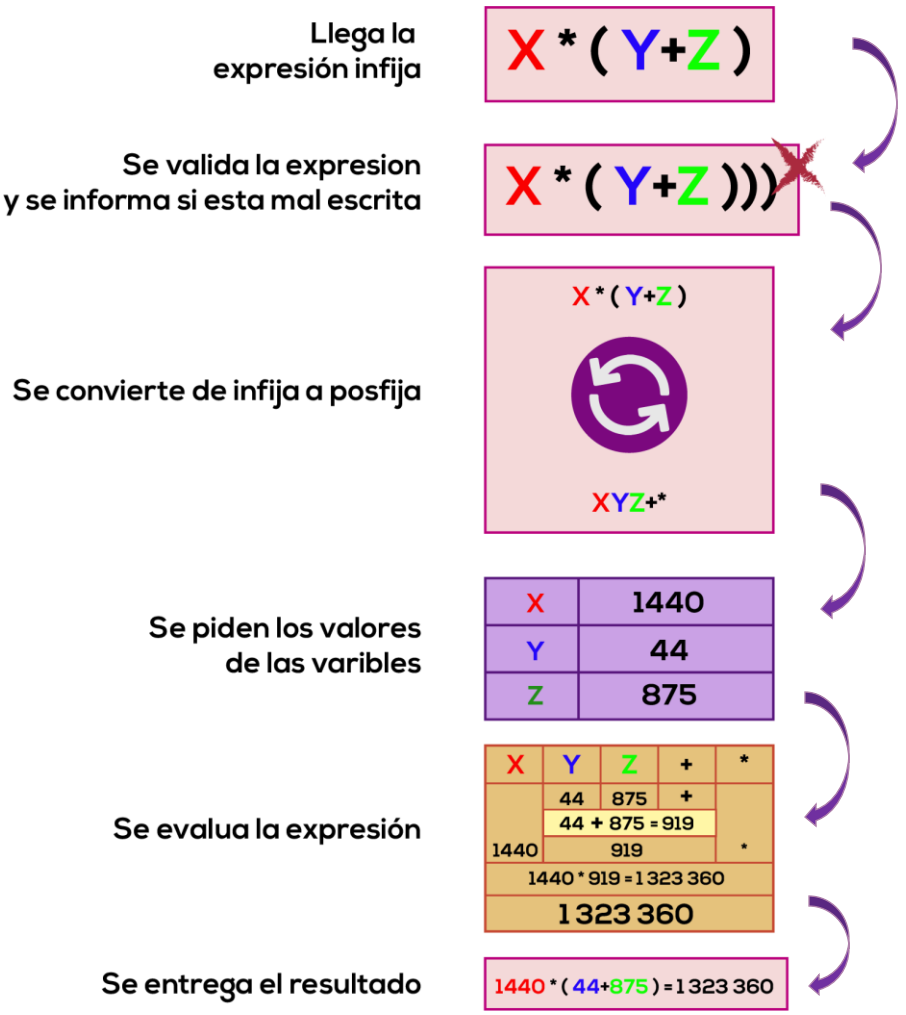
Planteamiento del problema

El problema principal a resolver en este planteamiento es la resolución de jerarquías que representan un problema regular en las expresiones matemáticas, y la complejidad para resolverlo, de manera que se tendrá que evaluar qué método usar para poder resolver la expresión sin necesidad de hacer uso de paréntesis, y poder resolver la ecuación de manera lineal.

Como problemas secundarios, nos encontramos con la escritura de la ecuación, la cual debe de ser validada para verificar que puede ser resuelta sin generar ningún problema en los procesos internos del programa; Igualmente se debe de evaluar la expresión (que estará dada en variables), para pedir una única ocasión los valores de dichas variables, para después ser sustituidas al momento de la resolución de la expresión.

Diseño de solución

Aquí se plantea un diagrama ordenado que muestra el diseño de la solución: [1]



Implementación de solución

La función *main*.

Dentro de la función *main* creamos los arreglos de caracteres en donde almacenaremos las expresiones que operaremos, además de arreglos de elementos auxiliares que nos facilitaran el uso de la librería TADPilaDin y TADPilaEst. También creamos: la pila en la cual haremos la conversión de la expresión infija a postfija, variables de tipo entero que nos servirán como contadores y un arreglo de enteros que contendrá los valores de las variables (A-Z).

Dentro de la función solicitamos al usuario que ingrese la expresión infija a transformar y la guardamos en un arreglo de caracteres para posteriormente agregarlo a un arreglo auxiliar (arreglo de *struct* elemento). Después llamamos a la función *cadParentesis*, la cual valida que la expresión tenga tanto los mismos paréntesis de apertura como los de cerrar.

```
int main(){
    int tam, i, jpos=0, valores[26];
    char cad[100], cadPos[100];
    elemento arrElemento[100], pos[100];
    float res;
    printf("Ingresa la expresion infija: ");
    scanf("%s", &cad);
    system("cls");
    printf("Expresion infija: %s\n", cad);
    tam = strlen(cad);
    for (i = 0; i < tam; ++i){
        arrElemento[i].c = cad[i];
    }


    if(cadParentesis(cad, tam) == 0){
        printf("ERROR: expresion mal escrita\n");
        exit(0);
    }

    pila p;
    Initialize(&p);
```

Luego, con un ciclo ***for,*** recorremos el arreglo auxiliar, el cual contiene la expresión infija a evaluar.

Primero abrimos un condicional donde llamamos a la función ***tipoOperador***, la cual nos devuelve un número entero del 0 - 4 que representa el tipo de operador (y si es mayor que 4 es una variable), que nos ayudara para clasificar los operadores y agregara directamente al carácter, en caso de que sea variable, al arreglo donde almacenamos la expresión postfija, Si el carácter es un operador, llamamos a la función ***agregarOp,*** la cual agregara el operador(s) a la expresión postfija siguiendo las reglas del nivel de presidencia, con ayuda de una pila. Finalmente, hacemos un ciclo ***while*** por si un operador quedo dentro de la pila.

```
for(i=0; i<tam; i++){
    if(tipoOperador(arrElemento[i]) < 5){
        agregarOp(arrElemento[i], &p, pos, &jpos, tipoOperador(arrElemento[i]));
    }
    else{
        pos[jpos] = arrElemento[i];
        jpos++;
    }
}


while(Empty(&p) == FALSE){
    pos[jpos] = Pop(&p);
    jpos++;
}
pos[jpos].c = '\0';
```

Ahora pasamos la expresión postfija, que habiamos almacenado en un arreglo de estructuras(***struct*** element), a un arreglo de caracteres; para poder imprimirlo en pantalla. Seguio de esto, hacemos uso de la función ***strupr***(incluida en la librería *string*.h) para convertir el arreglo de caracteres, donde almacenamos la expresión postfija, a mayusculas. Luego, inicializamos el arreglo en (-1) que contendra los valores de las variables(A-Z). Después, imprimimos en pantalla la expresión postfija, y llamamos a la función tablaValores, la cual pedira al usaurio los valores de las variables que amacenera en el arreglo ***valores***. Finalmente imprimimos el resultado de la expresión evaluada, llamando a la función resExpresion, la cual opera los valores de la expresión y devuelve el resultado.

```
    for(i=0; pos[i].c != '\0'; i++){
        cadPos[i] = pos[i].c;
    }
    cadPos[i] = '\0';

    strupr(cadPos);

    for(i=0; i<26; i++)
        valores[i] = -1;

    printf("Expresion postfijas: %s\n", cadPos);

    tablaValores(cadPos, valores);

    printf("\n\n---------------------------------------\nResultado: %.2f\n", resExpresion(cadPos, valores));
    Destroy(&p);
}
```

Función int cadParentesis(char cadena[], int tam_cadena).

Descripción: En esta función lo que se realiza es la verificación de la escritura de los paréntesis.

Recibe: *char* cadena[] (cadena dada por el usuario), *int* tam_cadena (tamaño de la cadena).

Devuelve: *int* (0 si hay un error, 1 la ejecución termina de manera correcta). [2]

```
int cadParentesis(char cadena[], int tam_cadena){
    int i,j;

    pila mi_pila;   //Se declara una pila "mi_pila"
    elemento e1;    //Declaro un elemento "e1"
    Initialize(&mi_pila);   //Inicialización de "mi_pila"

    for(i=0;i<tam_cadena;i++){  //Recorrer cada caracter de la cadena
        if(cadena[i]=='('){     //Si el caracter es ( introducirlo a la pila
            e1.c='(';
            Push(&mi_pila,e1);
        }
        else if(cadena[i]==')'){
            if(Empty(&mi_pila)) //Si se intenta extraer un elemento y la pila
                                //es vacia Error: P.g. (a+b)*c)
                return 0;
            e1=Pop(&mi_pila);
            }
    }
    //Si al terminar de revisar la expresión aún hay elementos en la pila
                                            //Error:P.g. (a+b)*c(a-c
    if(!Empty(&mi_pila))
        return 0;

    Destroy(&mi_pila);      //Si la ejecución termina de manera correcta
    return 1;
}
```

Función int tipoOperador(elemento e).

Descripción: Da la prioridad a los operadores y si está en el alfabeto A-Z

Recibe: elemento e (elemento a asignar prioridad).

Devuelve: int (prioridad del elemento dado).

```
int tipoOperador(elemento e){
    char op; //se declara op para asignarle que es un
             //elemento de tipo char y asi poder realizar la comparación correctamente.
    op = e.c;
    if(op == '-' || op == '+') //compara si el op es alguno de estos operadores y si se cumple retorna un 2.
        return 2;
    if(op == '*' || op == '/') //compara si el op es alguno de estos operadores y si se cumple retorna un 3.
        return 3;
    if( op == '^') //compara si el op es este operador y si se cumple retorna un 4.
        return 4;
    if(op == ')') //compara si el op es este operador y si se cumple retorna un 1.
        return 1;
    if(op == '(') //compara si el op es este operador y si se cumple retorna un 0.
        return 0;
    if(op>64 && op<91) ///compara si op esta dentro de este rango que es el rango de la tabla ASCII
                       // para las letras mayusculas y si se cumple retorna un 6.
        return 6;

    return 5;
}
```

Función void agreparOp(elemento op, pila *X, elemento pos[], int *j, int tipo).

Descripción: Agrega los operadores de forma correcta a la pila donde se encontrara la expresión posfija.

Recibe: elemento op (elemento que se ingresara a la pila de posfijo), pila *X (referencia a la pila), elemento *pos[]* (arreglo de elementos donde se guarda la expresión posfija), *int* *j (referencia al contador del arreglo de la expresion posfija), *int* tipo (tipo de operador que se va a ingresar a la pila).

Devuelve: un valor entero.

Observaciones: Dependiendo de la prioridad del elemento(s) realizara una acción, ya sea *Push* o *Pop.*

```
void agregarOp(elemento op, pila *X, elemento pos[], int *j, int tipo){
    switch(tipo){ //Se hace un switch con el valor de tipo, que se le asigno en la función tipoOperador.
        case 0:     //En caso de que sea 0, significa que es '(' por lo tanto
                    // se ingresa el operador, ya que '(' tiene precedencia 0.
            Push(X, op);
        break;
        case 1:     //En caso de que sea 1, significa que es ')' por lo tanto va a
                    //hacer el Pop, hasta que en el tope de la pila se encuentre un '('.
            while(tipoOperador(Top(X)) != 0){ //si encuentra a ")" vacia la pila.
                pos[*j] = Pop(X);
                (*j)++;
            }
            Pop(X); //Saca el '('
        break;
        case 2: case 3: case 4: //En cualquiera de los demas operadores se va a realizar un Push si
                                //la pila esta vacia o si el tipo de operador que se desea ingresar
                                //a la pila es mayor al tipo de operador que se encuentra en el tope de la pila.
            if(Empty(X) == TRUE || tipo > tipoOperador(Top(X))){
                Push(X, op);
            }else{      //Si no es ninguno de los casos anteriores, entonces se va a tener que realiza
                        // un pop, solo si la fila no esta vacia o si el tipo de operador que se desea
                        // ingresar es menor o igual al tipo de operador que se encuentra en el tope de la pila.
                while(Empty(X) == FALSE && tipo <= tipoOperador(Top(X))){
                    pos[*j] = Pop(X);
                    (*j)++;
                }
                Push(X, op);
            }
        break;
    }
}
```

Función int consultaValor(char c, int valores[]).

Descripción: Consulta el valor de los valores que se asignaron a las letras dadas por el usuario.

Recibe: char c(letra dada por el usuario), int valores[](valores que se desean consultar).

Devuelve: int i(valor consultado en la letra).

```
int consultaValor(char c, int valores[]){
    //consulta el valor exisntente en el arreglo de valores.
    return valores[c-'A'];
}
```

Función float resExpresion(char expPos[], int valores[]).

Descripción: Realiza la evaluación de la expresión posfija.

Recibe: char expPos[](cadena de la expresión postfija), int valores[](valores de las letras).

Devuelve: float (resultado de la evaluación postfija).

```
float resExpresion(char expPos[], int valores[]){
    int c,i; //Declaracion de contadores.
    elemento aux, val, operando1, operando2; //Declaraciones de variables tipo elemento
    pila numeros; //Declaracion de l pila que ayudara a la evaluacion de la expresion postfija
    Initialize(&numeros); //Inicializar pila.

    for(c=0; c<strlen(expPos); c++){ //Inicio del for que se realizara hasta que c
                                     //sea menor al tamaño de la cadena expPos
        aux.c = expPos[c]; //Se asigna a la variable auxiliar el valor de la expPos
                           //en la posicón de c.

        if(tipoOperador(aux) == 6){ //si el elemento que esta en aux es de tipo operador 6
                                    //significa que es una letra, por lo tanto entra a la condición.
            val.f = (float)consultaValor(expPos[c], valores); //val es una elemento de tipo
                                                              //flotante, se le asigna el valor que se consulta de la letra
                                                              //que esta en ese momento en expPos[c]
            Push(&numeros, val);        //Ya encontrado el valor deseado, se realiza
                                        //un push en la pila del valor de val.
        }
        if(tipoOperador(aux) < 5){ //si el elemento que esta en aux es de tipo
                                   //operador menor que 5 significa que es un operador,
                                   // por lo tanto entra dentro de la condicion.
            //Se realizara un pop de los dos ultimos operadores que estan en
            // la cima de la pila y se le asignaran a operador1 y operador2 respectivamente.
            operando1 = Pop(&numeros);
            operando2 = Pop(&numeros);
            //switch para realizar la operación que sea necesaria, donde a val.f se le asignara
            // el resultado de la operacion que se va a realizar.

        operando1 = Pop(&numeros);
        operando2 = Pop(&numeros);
        //switch para realizar la operación que sea necesaria, donde a val.f se le asignara
        // el resultado de la operacion que se va a realizar.
        switch(expPos[c]){
            case '+':
                val.f = operando1.f + operando2.f;
            break;

            case '-':
                val.f = operando2.f - operando1.f;
            break;

            case '*':
                val.f = operando1.f * operando2.f;
            break;

            case '/':
                val.f = operando2.f / operando1.f;
            break;

            case '^':
                val.f = pow(operando2.f, operando1.f);
            break;
        }

        Push(&numeros, val);
    }
}

    //Aqui se revisa si hay algun error, ya que si en la pila numeros, solo queda
    //un elemento el final siginifica que el proceso fue el correcto, pero si hay mas
    //de 1 significa que ocurrio algun error.
    if(Size(&numeros) > 1){
        system("cls");
        for(i=0; i<10; i++)
            system("@path && echo ---System protection permision denied---");
        printf("Hubo un error en la redaccion de tu expresion\n");
        return -1;
    }
    val = Pop(&numeros);
    return  val.f;
}
```

Función void tablaValores(char expPos[], int valores[]).

Descripción: Guarda los valores dados por el usuario.

Recibe: char expPos(cadena de expresión postfija), int valores[](campo en el cual se va a guardar el número asignado).

Devuelve: el resultado de la expresión postfija

Observaciones: Si la letra esta repetida no vuelve a pedir el valor.

```
void tablaValores(char expPos[], int valores[]){
    int c, valorTemp, pos;
    elemento aux;
    for(c=0; c<strlen(expPos); c++){ //Se inicia el for desde c=0 hasta que sea menor al tamaño de la cadena.
        aux.c = expPos[c];      //a aux se le asigna el valor que este en expPos en la posición de c
        if(tipoOperador(aux) == 6){ //Si el elemento que este en aux es igual a 6 significa que es una letra,
                                    // por lo tanto entra a la condición.
            pos = expPos[c] - 'A';; //a pos se le va a asignar la posicion que resulte al restar el valor ASCII
                                    //de la letra guardada en la expPos en la posición de c, menos el valor ASCII de A
            if(valores[pos] == -1){ //Si valores e -1 significa que NUNCA se le ha asignado un valor.
                printf("Introduce el valor de la variable %c: ", expPos[c]);
                fflush(stdin);
                scanf("%d", &valorTemp);
                valores[expPos[c] - 'A'] = valorTemp;
            }
        }
    }
}
```

Función de la solución

Que funcione sido o sea pantallazos de los resultados

Errores detectados

Dentro de los errores detectados en la entrega del programa, se encontró con el conflicto de que le programa no aceptaba como valores para sus variables, números de tipo decimales, esto debido a que al momento de declarar los tipos de variables primitivas no se tomó en cuenta el factor de que el usuario quisiera ingresar números decimales, sin embargo, para los resultados si se consideró la posibilidad de los resultados decimales. Por lo tanto, éste se considera un error de paralaje.

Durante la implementación del programa, si se encontraron diversos problemas, que tenían que ver la parte lógica del programa, como por ejemplo agregar valores enteros o flotantes en la realización de la pila, lo cual se solucionó agregando tipos de variables primitivas en el *struct* elemento de la librería .h de la pila.

Así como el anterior problema, se encontraron diversos errores, sin embargo, mediante un poco de análisis en equipo se logró llegar a las soluciones pertinentes que dieron pie a pocos errores a la hora de realizar los casos de prueba sobre el programa. Sin embargo, al no ser pruebas exhaustivas se tiene la posibilidad de detectar posibles errores en el futuro, de manera que habría que perfeccionarse el programa al detectarse dichos errores.

Posibles mejoras

Nosotros lo que hicimos al realizar el código fue ingresar los datos en una cadena de caracteres y luego pasar esa cadena a una cadena de elementos, lo cual nos complicó un poco la realización de algunas cosas, pero en algunas nos resultó más fácil, pero con esto se puso observar que se podría asignar una variable auxiliar del tipo elemento, para poder hacer las mismas operaciones, sin la necesidad de realizar los cambios mencionados anteriormente, así el programa gastaría menos memoria y mejoría la eficiencia.

Esto haría que existiera una disminución en el código, ya que nos ahorraríamos dos ciclos for para convertir la cadena de caracteres ingresados por el usuario y la cadena de caracteres de la expresión postfija.

Conclusiones

Azpeitia:

Fue una práctica que me dio una visión de lo potentes que pueden ser las estructuras de datos, como un simple arreglo de elementos puede hacer cosas muy complejas, para el desarrollo de software. La práctica nos mostró la utilidad y funcionamiento de una pila.

Gómez:

Personalmente me pareció una práctica excelente, ya que estimula la utilización de la estructura de datos Pila en un determinado modo que amplía el panorama de visión sobre cómo, cuándo y por qué se utilizará la Pila. A la vez me siento a gusto con mis compañeros de trabajo pues se notó una cooperación proactiva y mucho interés por averiguar cómo se lograría hacer funcionar el esquema de postfijo en una pila, y porque sería mucho más sencillo implementarlo con una pila que con otras maneras.

Raya:

Fue una práctica, que me ayudo a entender más sobre el funcionamiento de las pilas, y el cómo se pueden facilitar muchas cosas con el uso de la misma, que algunas cosas me costaron trabajo, pero al realizarlas de forma detallada y con la ayuda de mis compañeros de equipo me fueron más fáciles y claras.

Anexos

Todos los archivos fuente usados para la realizacion de este proyecto se pueden encontrar en el siguiente enlace: (El enlace tiene vigencia de 8/Dic/2017, fecha que concluye el periodo 2018/1)

https://correoipn-my.sharepoint.com/personal/egomezr1300_alumno_ipn_mx/_layouts/15/guestaccess.aspx?folderid=00444b08ade9042e8ac734f82e7535df8&authkey=AahpiZT-oCvIxJFFbDNK8Ac&expiration=2017-12-08T06%3a00%3a00.000Z

A continuacion, se presenta el código fuente del programa:

```
/*
AUTORES: Paola Raya Tolentino,
         Eduardo Gómez Rodriguez,
         Vladimir Azpeitia Hernández.
VERSION: 1.0
DESCRIPCIÓN: Con la implimentación del TAD Pila se implemento
un programa que valida y evalua una expresión infija.
Que toma en consideración la procedencia de los operadores,
realiza una evaluación de paréntesis escritos correctamente,
convierte la expresión a postfijo, mostrando el resultado
y finalmente realiza la evaluación de la expresión.
COMPILACIÓN: gcc -o posfija posfija.c TADPilaDin.o
             gcc -o posfija posfija.c TADPilaEst.o
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include "TADPilaDin.h"
#include "TADPilaEst.h"
#include <math.h>
/*
int cadParentesis(char cadena[], int tam_cadena);
Descripción: En esta función lo que se realiza es la verificación de la escritura
de los parentesis.
Recibe: char cadena[] (cadena dada por el usuario), int tam_cadena
(tamaño de la cadena).
Devuelve: int (0 si hay un error, 1 la ejecución termina de manera correcta).
*/
int cadParentesis(char cadena[], int tam_cadena){
    int i,j;

    pila mi_pila;   //Se declara una pila "mi_pila"
    elemento e1;    //Declaro un elemento "e1"
    Initialize(&mi_pila);   //Inicialización de "mi_pila"

    for(i=0;i<tam_cadena;i++){  //Recorrer cada caracter de la cadena
        if(cadena[i]=='('){     //Si el caracter es ( introducirlo a la pila
            e1.c='(';
            Push(&mi_pila,e1);
        }
        else if(cadena[i]==')'){
```

```
            if(Empty(&mi_pila)) //Si se intenta extraer un elemento y la pila es vacia Error:
P.g. (a+b)*c)
                    return 0;
                e1=Pop(&mi_pila);
            }
    }

    //Si al terminar de revisar la expresión aún hay elementos en la pila Error: P.g.
(a+b)*c(a-c
    if(!Empty(&mi_pila))
        return 0;

    //Si la ejecución termina de manera correcta
    Destroy(&mi_pila);
    return 1;
}

/*
int tipoOperador(elemento e);
Descripción: Da la prioridad a los operadores y si esta en el alfabeto A-Z
Recibe: elemento e (elemento a asignar prioridad).
Devuelve: int (prioridad del elemento dado).
*/
int tipoOperador(elemento e){
    char op; //se declara op para asignarle que es un elemento de tipo char y asi poder
realizar la comparación correctamente.
    op = e.c;
    if(op == '-' || op == '+') //compara si el op es alguno de estos operadores y si se
cumple retorna un 2.
        return 2;
    if(op == '*' || op == '/') //compara si el op es alguno de estos operadores y si se
cumple retorna un 3.
        return 3;
    if( op == '^') //compara si el op es este operador y si se cumple retorna un 4.
        return 4;
    if(op == ')') //compara si el op es este operador y si se cumple retorna un 1.
        return 1;
    if(op == '(') //compara si el op es este operador y si se cumple retorna un 0.
        return 0;
    if(op>64 && op<91) ///compara si op esta dentro de este rango que es el rango de la tabla
ASCII para las letras mayusculas y si se cumple retorna un 6.
        return 6;

    return 5;
}
/*
void agreparOp(elemento op, pila *X, elemento pos[], int *j, int tipo);
Descripción: Agrega los operadores de forma correcta a la pila
donde se encontrara la expresión posfija.
Recibe: elemento op (elemento que se ingresara a la pila de posfijo),
pila *X (referencia a la pila), elemento pos[]
(arreglo de elementos donde se guarda la expresión posfija),
int *j (referencia al contador del arreglo de la expresion posfija),
int tipo (tipo de operador que se va a ingresar a la pila).
Devuelve: i
Observaciones: Dependiendo de la prioridad del elemento
se realizara una accion, ya sea Push ó Pop;
*/
void agregarOp(elemento op, pila *X, elemento pos[], int *j, int tipo){
    switch(tipo){ //Se hace un switch con el valor de tipo, que se le asigno en la
función tipoOperador.
        case 0:     //En caso de que sea 0, significa que es '(' por lo tanto se
ingresa el operador, ya que '(' tiene precedencia 0.
```

```
            Push(X, op);
        break;
        case 1:     //En caso de que sea 1, significa que es ')' por lo tanto va a hacer el
Pop, hasta que en el tope de la pila se encuentre un '('.
            while(tipoOperador(Top(X) != 0){ //si encuentra a ")" vacia la pila.
                pos[*j] = Pop(X);
                (*j)++;
            }
            Pop(X); //Saca el '('
        break;
        case 2: case 3: case 4: //En cualquiera de los demas operadores se va a realizar un
Push si la pila esta vacia o si el tipo de operador que se desea ingresar a la pila es mayor
al tipo de operador que se encuentra en el tope de la pila.
            if(Empty(X) == TRUE || tipo > tipoOperador(Top(X))){
                Push(X, op);
            }else{      //Si no es ninguno de los casos anteriores, entonces se va a tener
que realizar un pop, solo si la fila no esta vacia o si el tipo de operador que se desea
ingresar es menor o igual al tipo de operador que se encuentra en el tope de la pila.
                while(Empty(X) == FALSE && tipo <= tipoOperador(Top(X))){
                    pos[*j] = Pop(X);
                    (*j)++;
                }
                Push(X, op);
            }
        break;
    }
}
/*
int consultaValor(char c, int valores[]);
Descripción: Consulta el valor de los valores que se asignaron
a las letras dadas por el usuario.
Recibe: char c(letra dada por el usuario),
int valores[](valores que se desean consultar).
Devuelve: int (valor consultado en la letra).
*/
int consultaValor(char c, int valores[]){
     //consulta el valor exisntente en el arreglo de valores.
    return valores[c-'A'];
}
/*
float resExpresion(char expPos[], int valores[]);
Descripción: Realiza la evaluación de la expresión posfija.
Recibe: char expPos[](cadena de la expresión postfia),
int valores[](valores de las letras).
Devuelve: float (resultado de la evaluación postfija).
*/
float resExpresion(char expPos[], int valores[]){
    int c,i; //Declaracion de contadores.
    elemento aux, val, operando1, operando2; //Declaraciones de variables tipo
elemento
    pila numeros; //Declaracion de l pila que ayudara a la evaluacion de la expresion
postfija
    Initialize(&numeros); //Inicializar pila.

    for(c=0; c<strlen(expPos); c++){ //Inicio del for que se realizara hasta que c
sea menor al tamaño de la cadena expPos
        aux.c = expPos[c]; //Se asigna a la variable auxiliar el valor de la expPos
en la posicón de c.

        if(tipoOperador(aux) == 6){ //si el elemento que esta en aux es de tipo
operador 6 significa que es una letra, por lo tanto entra a la condición.
```

```
            val.f = (float)consultaValor(expPos[c], valores); //val es una elemento de tipo
flotante, se le asigna el valor que se consulta de la letra que esta en ese momento en
expPos[c]
            Push(&numeros, val);        //Ya encontrado el valor deseado, se realiza un push
en la pila del valor de val.
        }
        if(tipoOperador(aux) < 5){ //si el elemento que esta en aux es de tipo operador menor
que 5 significa que es un operador, por lo tanto entra dentro de la condicion.
            //Se realizara un pop de los dos ultimos operadores que estan en la cima de la
pila y se le asignaran a operador1 y operador2 respectivamente.
            operando1 = Pop(&numeros);
            operando2 = Pop(&numeros);
            //switch para realizar la operación que sea necesaria, donde a val.f se le
asignara el resultado de la operacion que se va a realizar.
            switch(expPos[c]){
                case '+':
                    val.f = operando1.f + operando2.f;
                break;

                case '-':
                    val.f = operando2.f - operando1.f;
                break;

                case '*':
                    val.f = operando1.f * operando2.f;
                break;

                case '/':
                    val.f = operando2.f / operando1.f;
                break;

                case '^':
                    val.f = pow(operando2.f, operando1.f);
                break;
            }

            Push(&numeros, val);
        }
    }
    //Aqui se revisa si hay algun error, ya que si en la pila numeros, solo queda un elemento
el final siginifica que el proceso fue el correcto, pero si hay mas de 1 significa que
ocurrio algun error.
    if(Size(&numeros) > 1){
        system("cls");
        for(i=0; i<10; i++)
            system("@path && echo ---System protection permision denied---");
        printf("Hubo un error en la redaccion de tu expresion\n");
        return -1;
    }
    val = Pop(&numeros);
    return  val.f;
}
/*
void tablaValores(char expPos[], int valores[]);
Descripción: Guarda los valores dados por el usuario.
Recibe: char expPos(cadena de expresión postfija),
int valores[](campo en el cual se va a guardar el número asignado).
Devuelve:
Boservaciones: Si la letra esta repetida no vuelve a pedir
el valor.
*/
void tablaValores(char expPos[], int valores[]){
    int c, valorTemp, pos;
```

```
    elemento aux;
    for(c=0; c<strlen(expPos); c++){ //Se inicia el for desde c=0 hasta que sea menor al
tamaño de la cadena.
        aux.c = expPos[c];      //a aux se le asigna el valor que este en expPos en la
posición de c
        if(tipoOperador(aux) == 6){ //Si el elemento que este en aux es igual a 6 significa
que es una letra, por lo tanto entra a la condición.
            pos = expPos[c] - 'A';; //a pos se le va a asignar la posicion que resulte al
restar el valor ASCII de la letra guardada en la expPos en la posición de c, menos el valor
ASCII de A
            if(valores[pos] == -1){ //Si valores e -1 significa que NUNCA se le ha asignado
un valor.
                printf("Introduce el valor de la variable %c: ", expPos[c]);
                fflush(stdin);
                scanf("%d", &valorTemp);
                valores[expPos[c] - 'A'] = valorTemp;
            }
        }
    }
}
//PROGRAMA PRINCIPAL
int main(){
    //Declaracion de variables
    int k, m, tam, i, jpos=0, x=10, valores[26];
    char cad[100], cadPos[100];     //cad --> la que da el usuario. cadPos --> es donde se va
a escribir la expresión porfija
    elemento arrElemento[100], pos[100];        // arrElemento --> Arreglo de elementos donde
se copiara la cadena dada por el usuario, pos--> Arreglo donde se guardara la expresión
postfija.
    float res;      // Variable tipo flotante al que se le asignara el resultado de la
evaluación de la expresión dada.
    printf("Ingresa la expresion infija: ");
    scanf("%s", &cad);      //Se captura la cadena dada por el usuario.
    system("cls");      //Limpia la pantalla y vuelve a escribir la cadena dada por el
usuario.
    printf("Expresion infija: %s\n", cad);
    tam = strlen(cad);      //Guarda en tam el valor de la cadena da da por el usuario
    // printf("%d\n", tam);
    for (i = 0; i < tam; ++i){      //Se captura la cadena y se convierte a cadena de tipo
elemento.
        arrElemento[i].c = cad[i];
    }

    //Se valida la escritura de la cadena invocado a la función cadParentesis
    if(cadParentesis(cad, tam) == 0){ //pregunta si el valor que retorno la función
es cero, si sí es cero, es que hubo un error en la escritura de la función.
        printf("ERROR: expresion mal escrita\n");
        exit(0);
    }

    pila p;
    Initialize(&p); //Se inicializa pila para poderar operar sobre las prioridades de
los operandos.

    //Se evalua toda la funcion para convertirla a posfija
    for(i=0; i<tam; i++){       //se comienza un ciclo for con i=0 hasta que sea
menor al tamaño de la cadena ya y comienza a recorrer todo el arreglo verificando si
los elementos que estan dentro de ella son operadores o letras.
        if(tipoOperador(arrElemento[i]) < 5){
            agregarOp(arrElemento[i], &p, pos, &jpos, tipoOperador(arrElemento[i]));
//Si el elemento es operador se invoca a la función agregarOp, que agregara el
operador a la pila que se creo anteriormente.
        }
```

```
        else{ //Si el elemento es una letra se agrega automaticamente a la cadena pos. (que
es la cadena donde se guardara la expresión postfija)
            pos[jpos] = arrElemento[i];
            jpos++;
        }
    }

    while(Empty(&p) == FALSE){
        pos[jpos] = Pop(&p);
        jpos++;
    }
    pos[jpos].c = '\0';

    //Imprime la cadena posfija
    for(i=0; pos[i].c != '\0'; i++){
        cadPos[i] = pos[i].c;
    }
    cadPos[i] = '\0';

    strupr(cadPos); //Convierte los caracteres a mayusculas para no tener problema con cadPos

    for(i=0; i<26; i++)     //Asigna el valor -1 a todas las celdas del arreglo valores para
idnicar que no se le ha asignado ningun valor.
        valores[i] = -1;

    printf("Expresion postfijas: %s\n", cadPos);        //Imprime la expresión en postfijo.


    tablaValores(cadPos, valores);

    printf("\n\n---------------------------------------\nResultado: %.2f\n",
resExpresion(cadPos, valores));
    Destroy(&p);
}
```

Bibliografía

[1] E. G. Rodríguez, «Esquema de resolución para ecuaciones infijas,» S/E, Ciudad de México, 2017.

[2] ESCUELA SUPERIOR DE COMPUTO, ESTRUCTURAS DE DATOS, CIUDAD DE MEXICO, 2017.