INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Estructura de Datos

Práctica 1

Prof.: Franco Martínez Edgardo Adrián

Integrantes:

Mendoza Parra Sergio.

Salcedo Barrón Ruben Osmair.

Tejeda Martínez José Miguel.



MEXICO, D.F. a 21 de octubre del 2015

**Introducción:**

Haciendo uso de la implementación una el TAD Pila, en particular una estructura de datos estática (ya que se asigna una cantidad fija de memoria), se resolvió el problema planteado sobre la “Evaluación de expresiones Infijas”.

En este caso, se tomó en cuenta el tamaño máximo de la expresión, permitiendo tener más control sobre el programa funciones y demás. También se tomó en cuenta operaciones aritméticas con distintos niveles de precedencia.

Operaciones básicas:

^ Potencia

\*/ Multiplicación, división

+,- suma, resta.

Esto sirvió de mucho para poder tener bien la expresión postfija y poder determinar que usaríamos para la solución.

**Planteamiento del problema:**

Con la implementación del TAD Pila en C, (estática y dinámica) implementar un programa que valide y evalué una expresión infija. [1]

• Ejemplo

• A\*((B+D)/C)+E^A

1. Evaluación de paréntesis escritos correctamente

• El programa mostrara el resultado de la revisión de paréntesis

2. Conversión a posfijo y evaluación de la expresión posfija

• El programa mostrará el resultado de pasar la expresión a posfija y finalmente la evaluación de la expresión.

**Diseño y funcionamiento de la solución:**

**Conversión a postfija.**

Con base al programa que verificaba si la expresión dada tenia bien los ( ), tomamos en cuenta parte de esa implementación y le agregamos un arreglo de tamaño MAX\_ELEMENT, para así poder convertir la expresión sabiendo mediante el menú que la expresión era correcta y de este modo solo tener un caso donde solo se iba a convertir la expresión. Esto lo hicimos separando los operandos y las letras para poder pasarlas a la expresión postfija, recorriendo toda la expresión y metiendo a la pila los paréntesis y operandos para que así solo quedaran las letras y se acomodaran en un arreglo.

Ejemplo:

|  |
| --- |
|  |
| )   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |
| + |
| ( |

(A+B)/C-F

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** |  |  |  |  |

Aquí lo que queremos dar a entender es que cuando se recorre la cadena solo los operandos y paréntesis se meten a la pila y al momento de encontrar un “)” saca lo que halla en la pila y lo mete a la expresión el cual quedaría de la siguiente manera.

|  |
| --- |
|  |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |
|  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **+** |  |  |  |

Aquí la pila queda vacía y sigue recorriendo la expresión, así seguirá hasta que ya no haya nada en la pila y se termine de recorrer la cadena.

|  |
| --- |
|  |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |
|  |
| - |

(A+B)/C-F

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **+** | **C** |  |  |

A l intentar meter el – y haber un / antes en la pila por la precedencia se saca el / y se pone en el arreglo y en la pila solo queda el -.

Entonces el arreglo y la pila quedan de la siguiente manera:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **+** | **C** | **/** | **F** |  |

|  |
| --- |
|  |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |
| - |
| / |

Después de acomodar los operandos mediante la precedencia y una vez recorrida toda la expresión se saca todo de la pila y se agrega al arreglo, quedando de esta forma el arreglo:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **+** | **C** | **/** | **F** | **-** |

Ya que al no haber expresión se saca todo de la pila y se agrega al arreglo.

Este ejemplo es sencillo, esto para poder explicar de manera concisa el método donde acomodamos la expresión a postfija.

**Asignación de valores:**

Una vez convertida la expresión se hace un re-corrimiento de la cadena y si esta entre el rango de “A” a “Z” se pregunta por el valor de cada variable, después se hace un procedimiento similar para acomodar la expresión de tal manera que cuando se le asignen valores a la expresión la maquina pueda realizar las operaciones.

Ejemplo:

A=5, B=3, C=2, F=1.

AB+C/F-

5 3+ 2/ 1-

Al recorrer la cadena y encontrar en este caso un + y sabiendo que para que una expresión haga una operación debe de haber 2 números a su izquierda para esto se hace un pop de los números (e1 = Pop(&mi\_pila); y e2= Pop(&mi\_pila);)y así se hace la operación:

Suma=e2.d + e1.d;

|  |
| --- |
|  |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |
| 3 |
| 5 |

Y esto se guarda en la expresión para evaluar los siguientes operandos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **8** | **2** | **/** | **1** | **-** |  |

Siguiendo este procedimiento:

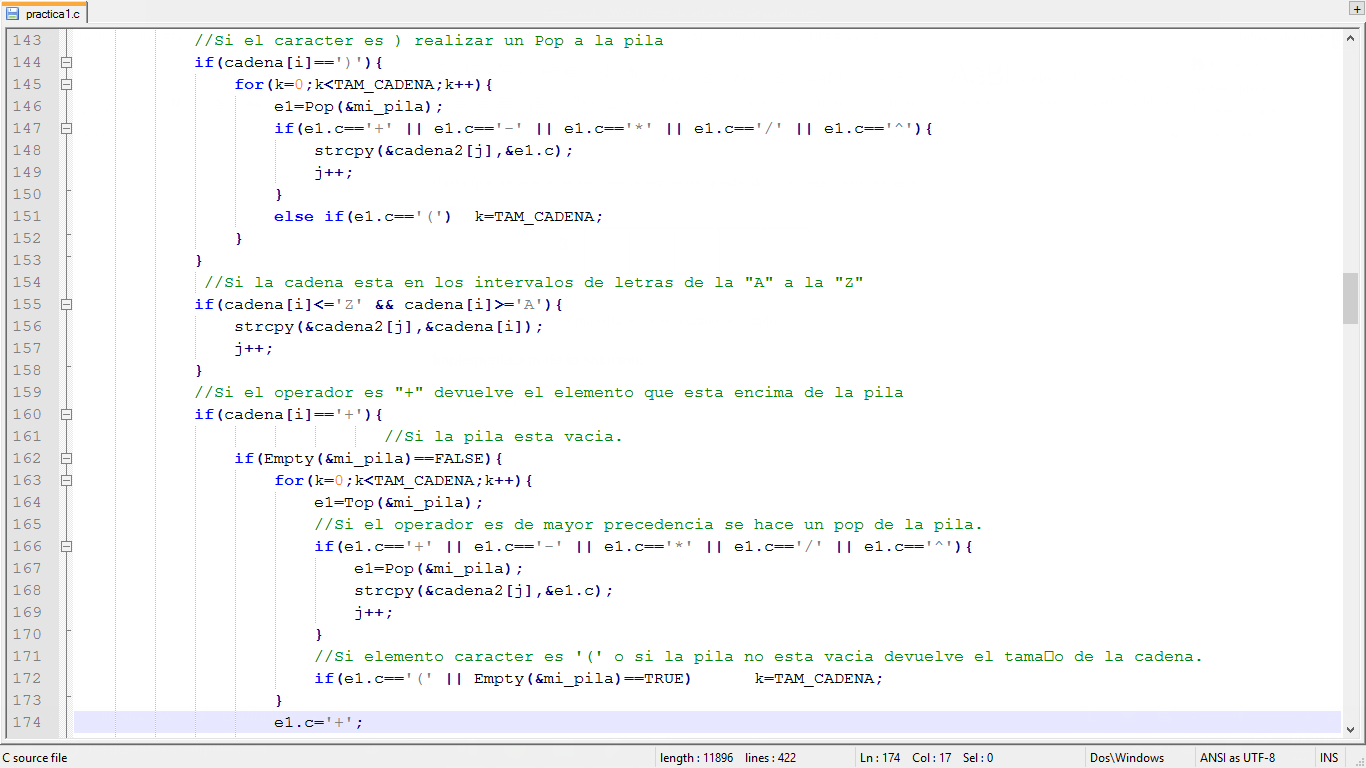
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **4** | **1** | **-** |  |  |  |

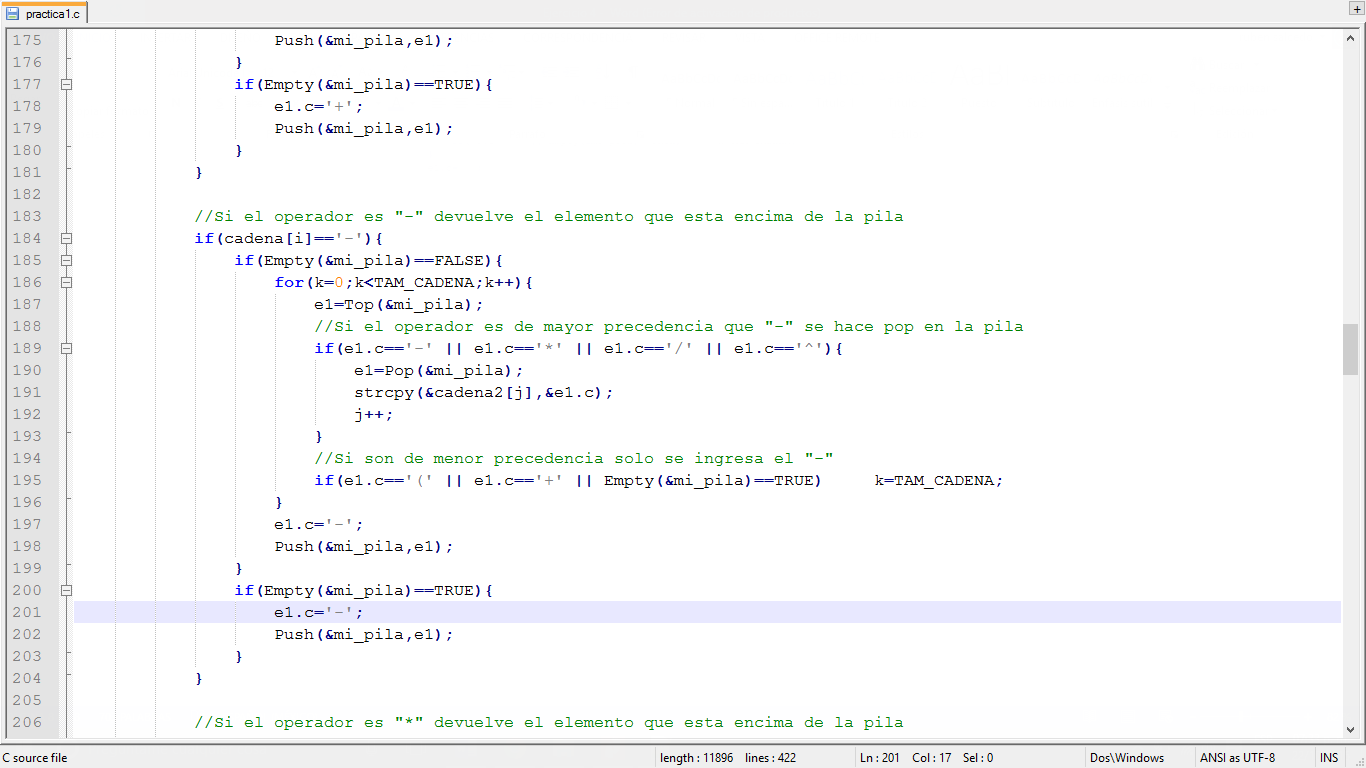
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **3** |  |  |  |  |  |

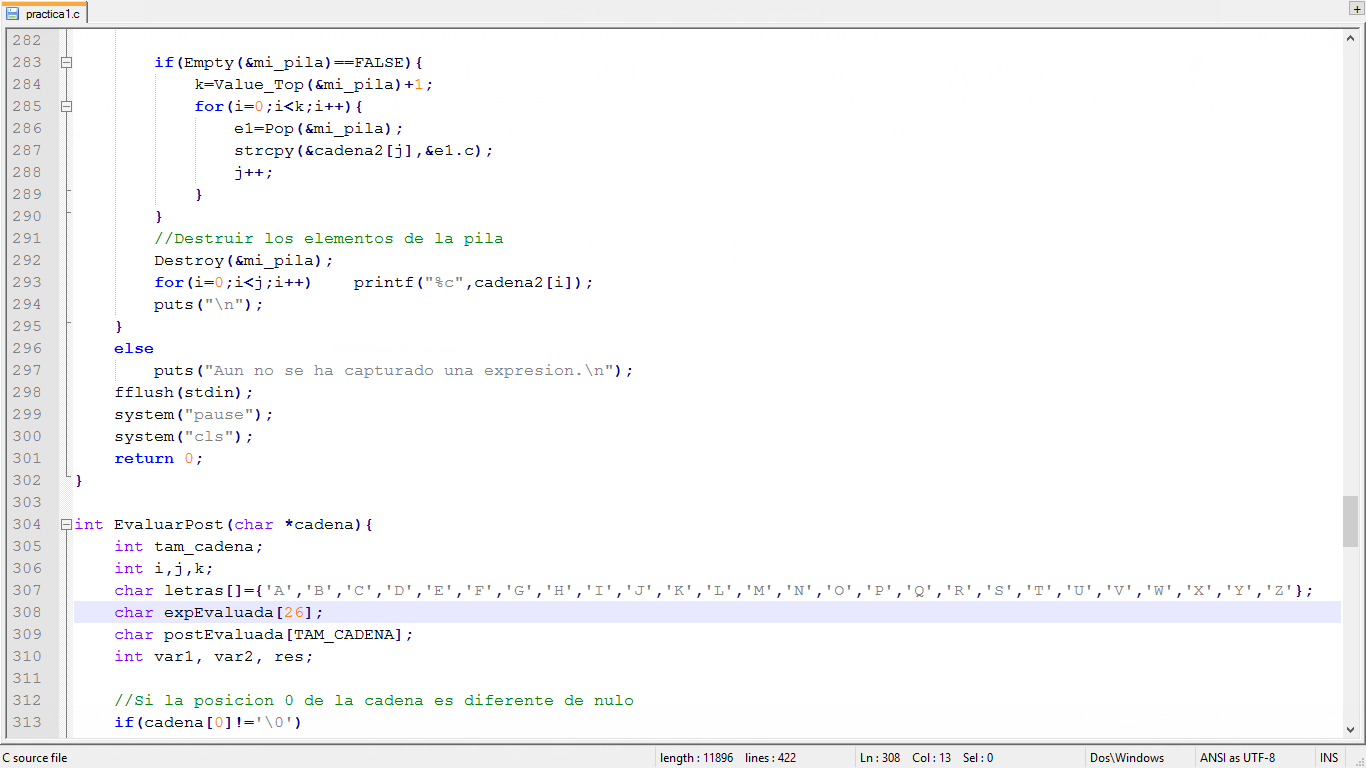
Hasta que se acabe de recorrer la expresión quedando como resultado (en este caso) así:

De esta manera solo se muestra la cadena como resultado.

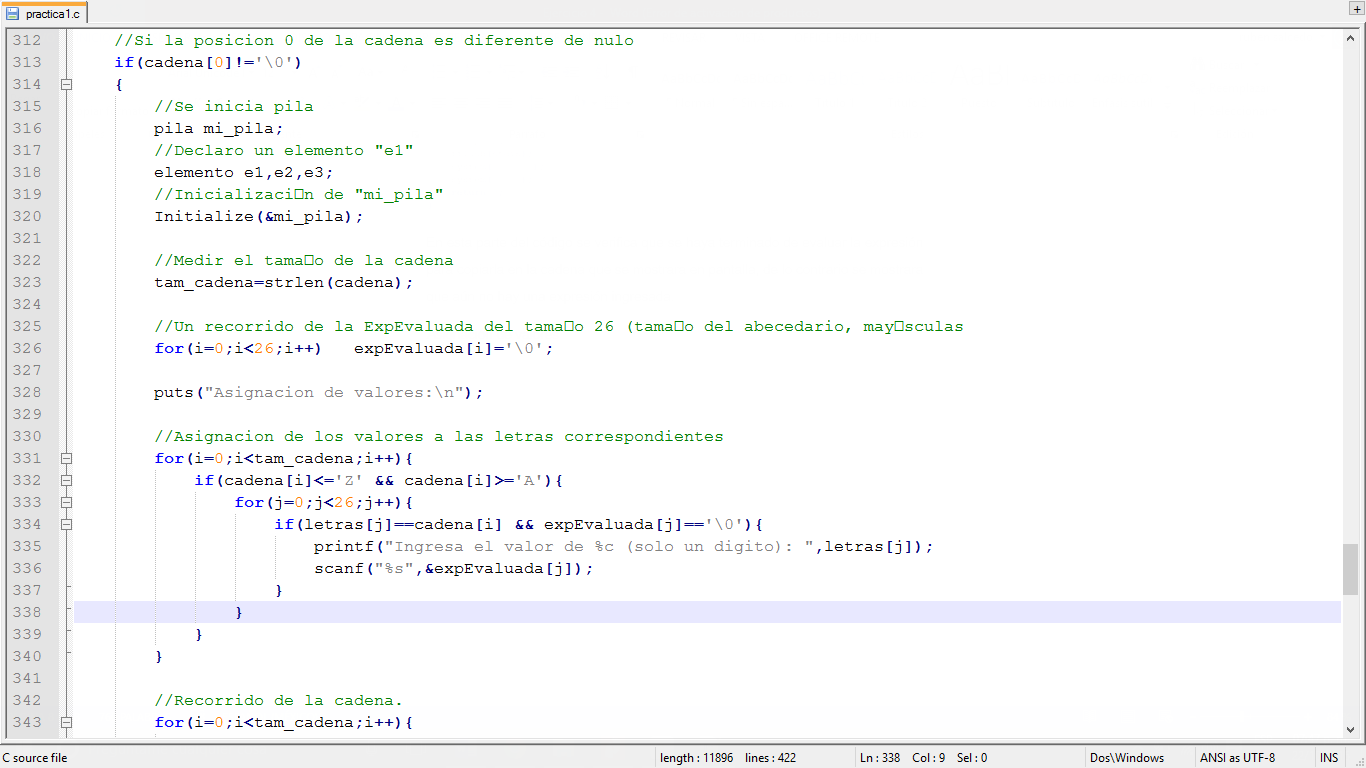
**Implementación de la solución:**

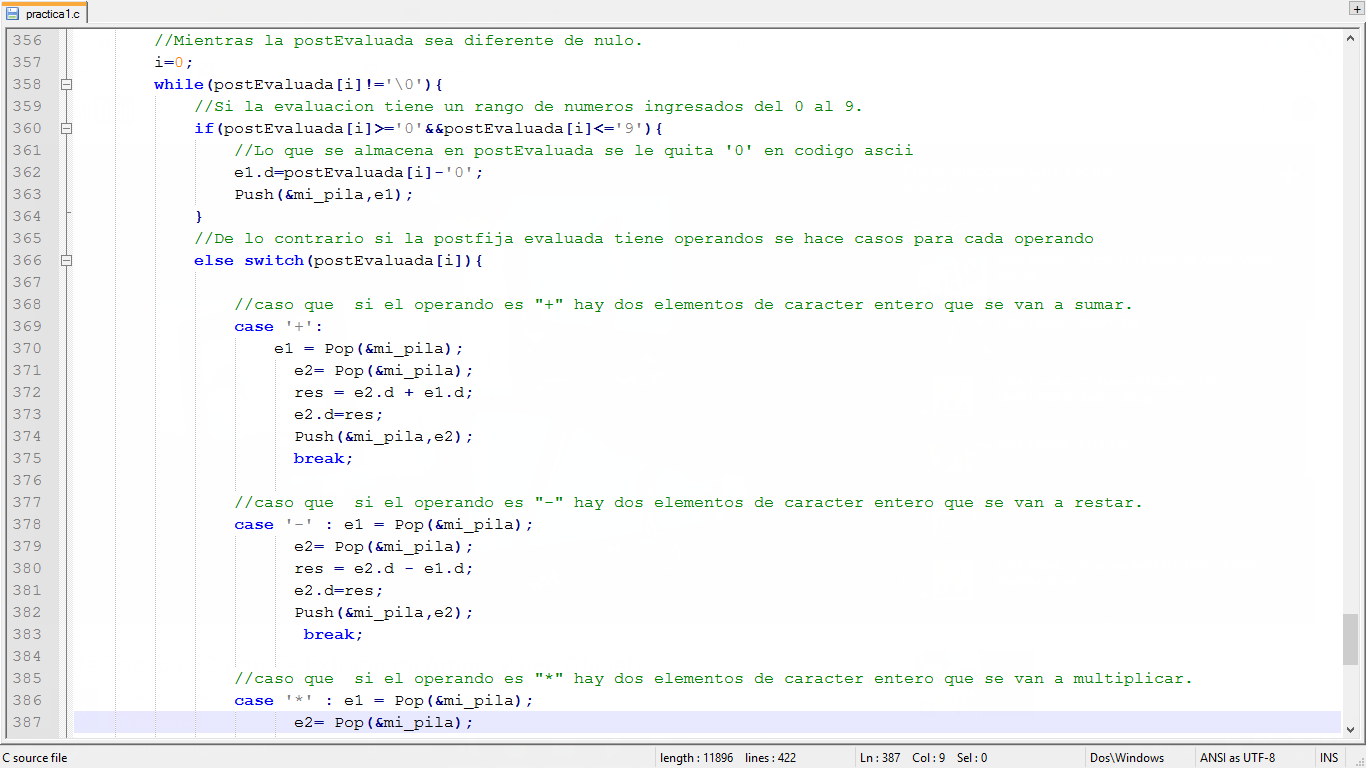
**Conversión a postfija**

En el código se observa el procedimiento para la + que es similar a los demás operandos.

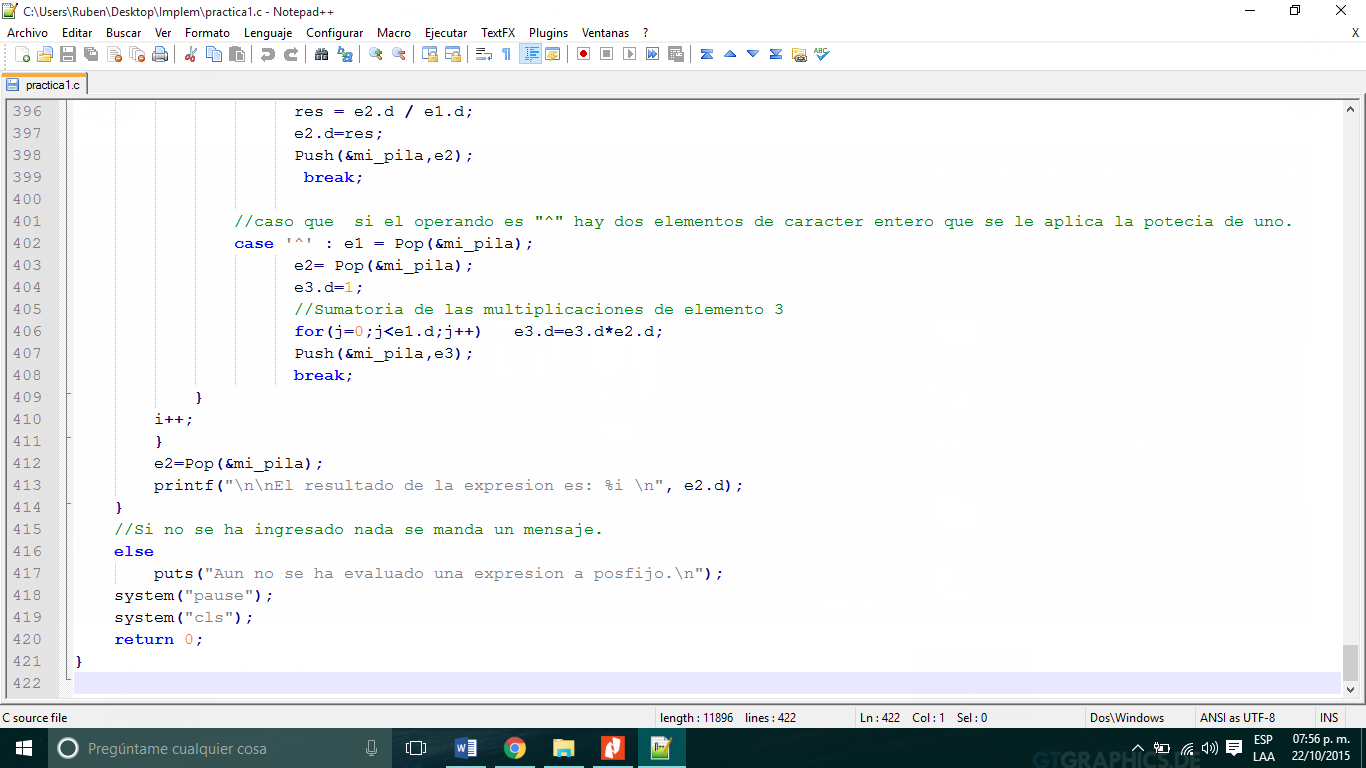


En esta parte del código se verifica que se haya terminado de evaluar la expresión para copiarla en la cadena que se mostrara en pantalla, de lo contrario se mostrara que aún no hay una expresión ingresada.

**Asignación de valores**

En esta para se vuelve a verificar si hay una expresión para poder realizar los cálculos, aquí se recorre la cadena para pedir los valores de las variables.

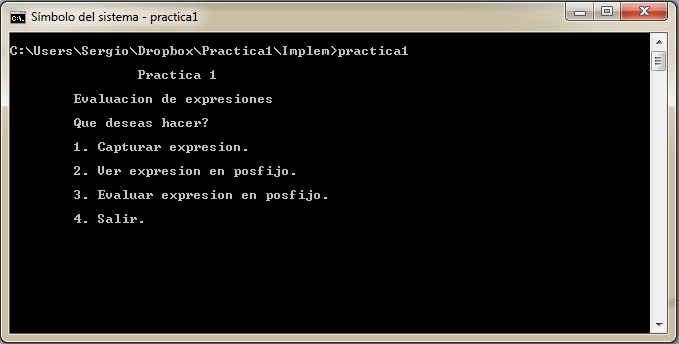
Una vez asignado los valores de las variables se hacen las operaciones según el operando. Y una vez hecho los cálculos se muestra en pantalla.

}

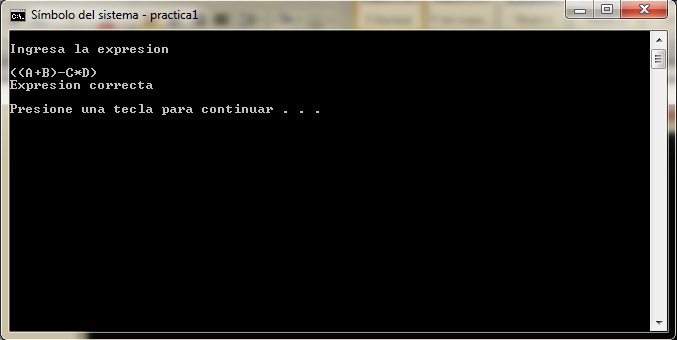
**Funcionamiento**

Funcionamiento del programa (Menú):

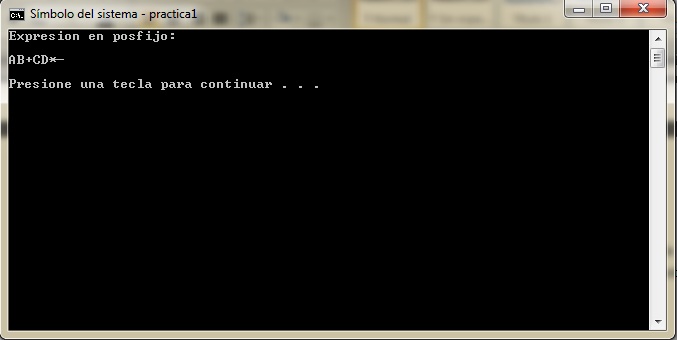
Aquí se puede observar la pantalla de salida del programa en forma de menú, la cual al momento de que el usuario teclee un número lo llevara a cada opción del menú, pero ingresando primero la expresión infija, ya que de lo contrario saldrá “Error”.



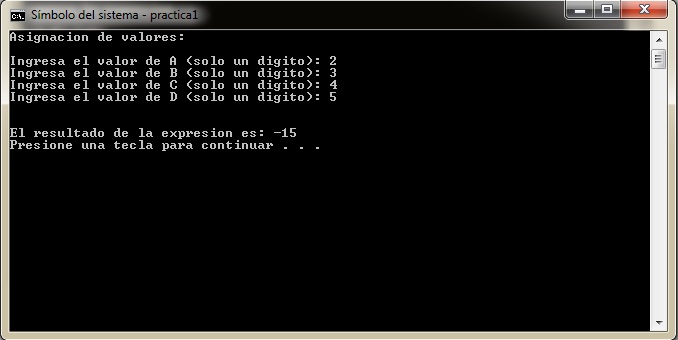
Después de haber ingresado la primera opción se ingresa la expresión en infijo, corroborando que los paréntesis estén con su respectivo paréntesis de cierre, que a su vez será evaluada una vez tecleando el botón de “enter”.



Una vez ingresado correctamente la expresión en infijo en el menú se podrá escoger tanto la “opción 2 o la opción 3” si queremos escoger la opción “2” nos aparecerá la expresión en postfijo.



Una vez ya visto la expresión en postfijo, por último se tendría que evaluar la expresión postfija con cualquier número que desee el usuario.



Se puede verificar el resultado ya que de la expresión infija tenemos:

Y por último el programa se regresa al menú para que el usuario si gusta pueda ingresar otra función en infijo y hacer lo mismo cuantas veces quiera o hasta que selecciona la opción de salida del programa.

**Errores detectados**

Los errores detectados fueron 2 los cuales fueron:

1. Algunas operaciones las hace bien y otras no: Fue un problema que se tuvo ya que si poníamos una expresión larga el programa a veces hacia mal las operaciones y se desconoció porque hacia mal las operaciones, nosotros creemos que fue en la condición de la potencia ya que tuvimos un poco de problemas ya que al momento de correr el programa nos salía error en el operador de elevar a una potencia y creemos que ese es el problema.
2. No se pudo ingresar más de un digito al momento de evaluar: Fue un problema que tuvimos y no se pudo resolver, ya que como pusimos que esos números se ingresaran en una cadena en el programa, al momento de ingresar por ejemplo “1.56” el programa ponía cada número en distinta posición en vez de que todo lo tomara en una sola posición, porque cuando corrimos nuestro programa al momento de realizar las operaciones, el programa hacia mal las cuentas y no salía el resultado que se tenía que dar y por ese motivo solo le pusimos un intervalo de “0<a<9”.

**Posibles mejoras**

Se puede disminuir el código:

1. No declarando doble vez la funciones que se van a usar en el programa ya que con solo utilizarlas en cada modulo de código es más que suficiente.
2. Otra posible disminución de código podría ser ingresar un “do – while” en función de int ConvertirPost(char \*cadena,char \*cadena2) , esto es para no poner tanta condición “if” en cada precedencia de los operadores del programa y así tener un poco mas de orden en lo que se está haciendo.

Algo que se podría implementar seria:

1. Que de una expresión en postfijo se convirtiera a infijo, lo cual sería lo mismo que hace el programa solo que al revés.

Algunas posibles soluciones de problemas que se tuvieron:

1. Podría ser que el número que ingrese el usuario se lea en forma de gets (en Windows) o getchar en (linux) porque como leen cadenas se podría poner en una cadena para que se pueda hacer la evaluación con números de más de 2 dígitos.

**Conclusiones**

Mendoza Parra Sergio: Fue una práctica interesante la verdad nos llevo tiempo pensarle hubo ocasiones en la que no se sabíamos por donde hacerle al programa, para mí la función más difícil fue en la que se tenía que hacer la conversión a postfijo, ya que se tenían que validar cada operando y cada condición dependiendo si había un operador de mayor precedencia o no, por lo que hubo algunas situaciones de confusión y como nos ayudábamos en equipo mientras uno tenía problemas otro le ayuda y así hasta que saliera el programa.

Salcedo Barrón Rubén Osmair: En esta práctica se usó el manejo de varias pilas para poder hacer más efectivo la solución del problema al igual que arreglos y sintaxis básica. Al principio de esta práctica nos costó un poco entender cómo funcionaba la pila para poder usarla y convertir la expresión, sin embargo nos habíamos limitado a solo pensar en pilas, pero después usamos arreglos para juntar las expresiones y hacerlo de maneras más fácil. También donde se nos complico fue al asignarle valores a las variables ya que no pudimos darle valores de más de 1 digito. Pero a pesar de eso quedo funcional el programa hasta cierto punto.

Tejeda Martínez José Miguel: Con base al problema planteado, tuvimos que hacer uso del TAD Pila proporcionar la solución. Después de haber realizado todo lo anterior, este tipo de estructura nos ayudó desde la parte de evaluación de los paréntesis [2] hasta los resultados parciales que se iban almacenando al momento de hacer la evaluación de la expresión. A pesar de haber encontrado errores muy básicos en nuestro programa (como la limitación para capturar valores de más de una cifra o números fraccionarios), puedo concluir que hacer uso tanto de las estructuras de control básicas como de los el TAD Pila es imprescindible para abstraer procesos que de igual forma los podemos hacer en el mundo real.

* Anexos(Códigos fuente)

El siguiente código muestra la totalidad de las líneas usadas para la resolución del problema con lo respectiva documentación.

/\*

Autores: Mendoza Parra Sergio, Salcedo Barrón Rubén Osmair, Tejeda Martínez José Miguel

Versión 1.0 (20 de Octubre de 2015)

Descripción: Programa que evalúa expresiones infijas

Con la implementación del TAD Pila el programa evalúa una expresión infija

1) El programa verifica la correcta escritura de paréntesis

2) El programa muestra la conversión de la expresión a posfijo

3) El programa permite la evaluación de la expresión posfija

Observaciones: El programa requerirá de la librería "TADPilaEst.h", la cual tiene las implementaciones para hacer uso de las funciones del TAD Pila

Compilación: gcc -o practica1 practica1.c TADPilEst.o "Si se tiene el código objeto"

gcc -o practica1 practica1.c TADPilaEst.c "Si se tiene el código fuente"

Ejecución: Windows practica1.exe & Linux ./practica1

\*/

//LIBRERIAS

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include "TADPilaEst.h"

//Tamaño máximo de la cadena

#define TAM\_CADENA 100

//Declaración de funciones

int Capturar(char \*cadena); //Función que se llama desde el menú principal para leer la cadena y evaluarla con la función EvaluaParentesis()

int EvaluaParentesis(char \*cadena); //Recibe la cadena evaluando que haga uso correcto de los paréntesis y se encuentre redactada correctamente

int ConvertirPost(char \*cadena,char \*cadena2); //Recibe la cadena anteriormente evaluada y realiza la conversión a posfijo, la cual de devuelve en "cadena2"

int EvaluarPost(char \*cadena); //Usando la cadena en posfijo se reemplazan los valores de las letras y realiza su evaluación para imprimir en pantalla

void main (void){

char expInfija[TAM\_CADENA], expPost[TAM\_CADENA];

int opc; //Opciones del menú

expInfija[0]='\0';

expPost[0]='\0';

do{

opc=0;

puts("\n\t\tPractica 1\n\n\tEvaluacion de expresiones");

puts("\n\tQue deseas hacer?\n");

puts("\t1. Capturar expresion.\n");

puts("\t2. Ver expresion en posfijo.\n");

puts("\t3. Evaluar expresion en posfijo.\n");

puts("\t4. Salir.");

fflush(stdin);

scanf("%d",&opc);

system("cls");

switch(opc){

case 1:

Capturar(expInfija);

break;

case 2:

ConvertirPost(expInfija,expPost);

break;

case 3:

EvaluarPost(expPost);

break;

case 4:

;

}

}

while(opc!=4);

return;

}

int EvaluaParentesis(char \*cadena)

{

int tam\_cadena;

int i;

//Se declara una pila "mi\_pila"

pila mi\_pila;

//Declaro un elemento "e1"

elemento e1;

//Inicializaci�n de "mi\_pila"

Initialize(&mi\_pila);

//Medir el tama�o de la cadena

tam\_cadena=strlen(cadena);

//Recorrer cada caracter de la cadena

for(i=0;i<tam\_cadena;i++)

{

//Si el caracter es ( introducirlo a la pila

if(cadena[i]=='(')

{

e1.c='(';

Push(&mi\_pila,e1);

}

//Si el caracter es ) realizar un Pop a la pila

else if(cadena[i]==')')

{

//Si se intenta extraer un elemento y la pila es vacía Error: P.g. (a+b)\*c)

if(Empty(&mi\_pila))

{

printf("ERROR: Existen mas parentesis que cierran de los que abren");

return (-1);//Salir del programa con error

}

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))

{

printf("ERROR: Existen mas parentesis que abren de los que cierran");

return (-1); //Salir del programa con error

}

//Si la ejecución termina de manera correcta

printf("Expresion correcta");

//Destruir los elementos de la pila

Destroy(&mi\_pila);

return 0;

}

int Capturar(char \*cadena)

{

do{

puts("\nIngresa la expresion\n");

scanf("%s",cadena);

}

while(EvaluaParentesis(cadena)==(-1));

puts("\n");

system("pause");

system("cls");

return 0;

}

int ConvertirPost(char \*cadena,char \*cadena2){

int tam\_cadena;

int i,j=0,k;

if(cadena[0]!='\0'){

//Se declara una pila "mi\_pila"

pila mi\_pila;

//Declaro un elemento "e1"

elemento e1;

//Inicializaci�n de "mi\_pila"

Initialize(&mi\_pila);

//Medir el tama�o de la cadena

tam\_cadena=strlen(cadena);

puts("Expresion en posfijo:\n");

//Recorrer cada caracter de la cadena

for(i=0;i<tam\_cadena;i++)

{

//Si el caracter es ( introducirlo a la pila

if(cadena[i]=='(')

{

e1.c='(';

Push(&mi\_pila,e1);

}

//Si el caracter es ) realizar un Pop a la pila

if(cadena[i]==')'){

for(k=0;k<TAM\_CADENA;k++){

e1=Pop(&mi\_pila);

if(e1.c=='+' || e1.c=='-' || e1.c=='\*' || e1.c=='/' || e1.c=='^'){

strcpy(&cadena2[j],&e1.c);

j++;

}

else if(e1.c=='(') k=TAM\_CADENA;

}

}

//Si la cadena esta en los intervalos de letras de la "A" a la "Z"

if(cadena[i]<='Z' && cadena[i]>='A'){

strcpy(&cadena2[j],&cadena[i]);

j++;

}

//Si el operador es "+" devuelve el elemento que esta encima de la pila

if(cadena[i]=='+'){

//Si la pila esta vacia.

if(Empty(&mi\_pila)==FALSE){

for(k=0;k<TAM\_CADENA;k++){

e1=Top(&mi\_pila);

//Si el operador es de mayor precedencia se hace un pop de la pila.

if(e1.c=='+' || e1.c=='-' || e1.c=='\*' || e1.c=='/' || e1.c=='^'){

e1=Pop(&mi\_pila);

strcpy(&cadena2[j],&e1.c);

j++;

}

//Si elemento caracter es '(' o si la pila no esta vac+ia devuelve el tama�o de la cadena.

if(e1.c=='(' || Empty(&mi\_pila)==TRUE) k=TAM\_CADENA;

}

e1.c='+';

Push(&mi\_pila,e1);

}

if(Empty(&mi\_pila)==TRUE){

e1.c='+';

Push(&mi\_pila,e1);

}

}

//Si el operador es "-" devuelve el elemento que esta encima de la pila

if(cadena[i]=='-'){

if(Empty(&mi\_pila)==FALSE){

for(k=0;k<TAM\_CADENA;k++){

e1=Top(&mi\_pila);

//Si el operador es de mayor precedencia que "-" se hace pop en la pila

if(e1.c=='-' || e1.c=='\*' || e1.c=='/' || e1.c=='^'){

e1=Pop(&mi\_pila);

strcpy(&cadena2[j],&e1.c);

j++;

}

//Si son de menor precedencia solo se ingresa el "-"

if(e1.c=='(' || e1.c=='+' || Empty(&mi\_pila)==TRUE) k=TAM\_CADENA;

}

e1.c='-';

Push(&mi\_pila,e1);

}

if(Empty(&mi\_pila)==TRUE){

e1.c='-';

Push(&mi\_pila,e1);

}

}

//Si el operador es "\*" devuelve el elemento que esta encima de la pila

if(cadena[i]=='\*'){

if(Empty(&mi\_pila)==FALSE){

for(k=0;k<TAM\_CADENA;k++){

e1=Top(&mi\_pila);

//Si los elementos igual o mayor precedencia

if(e1.c=='\*' || e1.c=='/' || e1.c=='^'){

e1=Pop(&mi\_pila);

strcpy(&cadena2[j],&e1.c);

j++;

}

//Si el elemento es diferente se ingresa el "\*"

if(e1.c!='(' ||e1.c!='+' || e1.c!='-' || Empty(&mi\_pila)==TRUE) k=TAM\_CADENA;

}

e1.c='\*';

Push(&mi\_pila,e1);

}

//Si la pila esta vacia solo se devuelve el operando

if(Empty(&mi\_pila)==TRUE){

e1.c='\*';

Push(&mi\_pila,e1);

}

}

//Si el operador es "/" devuelve el elemento que esta encima de la pila

if(cadena[i]=='/'){

if(Empty(&mi\_pila)==FALSE){

//Recorrido de la cadena para encontrar operadores de mayor precedencia

for(k=0;k<TAM\_CADENA;k++){

e1=Top(&mi\_pila);

//Si son de mayor o igual precedencia se saca el elemento

if(e1.c=='/' || e1.c=='^'){

e1=Pop(&mi\_pila);

strcpy(&cadena2[j],&e1.c);

j++;

}

//Si es de menor precedencia se queda en la pila el operador de mayor precedencia.

if(e1.c!='(' ||e1.c!='+' || e1.c!='-' || e1.c!='\*' || Empty(&mi\_pila)==TRUE)

k=TAM\_CADENA;

}

e1.c='/';

Push(&mi\_pila,e1);

}

//Si solo hay un operando se devuelve el mismo

if(Empty(&mi\_pila)==TRUE){

e1.c='/';

Push(&mi\_pila,e1);

}

}

//Si el operador es "^" devuelve el elemento que esta encima de la pila

else if(cadena[i]=='^'){

if(Empty(&mi\_pila)==FALSE){

//Recorrido de la cadena.

for(k=0;k<TAM\_CADENA;k++){

e1=Top(&mi\_pila);

//Si "^" es igual a su mismo operador

if(e1.c=='^'){

e1=Pop(&mi\_pila);

strcpy(&cadena2[j],&e1.c);

j++;

}

//Si es de menor precedencia se ingresa a la pila "^"

if(e1.c!='(' ||e1.c!='+' || e1.c!='-' || e1.c!='\*' || e1.c!='/' || Empty(&mi\_pila)==TRUE)

k=TAM\_CADENA;

}

e1.c='^';

Push(&mi\_pila,e1);

}

//Si es el único operando se devuelve el mismo

if(Empty(&mi\_pila)==TRUE){

e1.c='^';

Push(&mi\_pila,e1);

}

}

}

if(Empty(&mi\_pila)==FALSE){

k=Value\_Top(&mi\_pila)+1;

for(i=0;i<k;i++){

e1=Pop(&mi\_pila);

strcpy(&cadena2[j],&e1.c);

j++;

}

}

//Destruir los elementos de la pila

Destroy(&mi\_pila);

for(i=0;i<j;i++) printf("%c",cadena2[i]);

puts("\n");

}

else

puts("Aun no se ha capturado una expresion.\n");

fflush(stdin);

system("pause");

system("cls");

return 0;

}

int EvaluarPost(char \*cadena){

int tam\_cadena;

int i,j,k;

char letras[]={'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

char expEvaluada[26];

char postEvaluada[TAM\_CADENA];

int var1, var2, res;

//Si la posición 0 de la cadena es diferente de nulo

if(cadena[0]!='\0')

{

//Se inicia pila

pila mi\_pila;

//Declaro un elemento "e1"

elemento e1,e2,e3;

//Inicializaci�n de "mi\_pila"

Initialize(&mi\_pila);

//Medir el tama�o de la cadena

tam\_cadena=strlen(cadena);

//Un recorrido de la ExpEvaluada del tamaño 26 (tamaño del abecedario, mayúsculas

for(i=0;i<26;i++) expEvaluada[i]='\0';

puts("Asignacion de valores:\n");

//Asignación de los valores a las letras correspondientes

for(i=0;i<tam\_cadena;i++){

if(cadena[i]<='Z' && cadena[i]>='A'){

for(j=0;j<26;j++){

if(letras[j]==cadena[i] && expEvaluada[j]=='\0'){

printf("Ingresa el valor de %c (solo un digito): ",letras[j]);

scanf("%s",&expEvaluada[j]);

}

}

}

}

//Recorrido de la cadena.

for(i=0;i<tam\_cadena;i++){

//Si están en el intervalo del "A" a la "Z"(mayúsculas)

if(cadena[i]<='Z' && cadena[i]>='A'){

for(j=0;j<26;j++){

//Si alguna letra es encontrada, se copia en la posición "0" de letras[j].

if(letras[j]==cadena[i]){

strcpy(&postEvaluada[i],&expEvaluada[j]);

}

}

}

else strcpy(&postEvaluada[i],&cadena[i]);

}

//Mientras la postEvaluada sea diferente de nulo.

i=0;

while(postEvaluada[i]!='\0'){

//Si la evaluacion tiene un rango de números ingresados del 0 al 9.

if(postEvaluada[i]>='0'&&postEvaluada[i]<='9'){

//Lo que se almacena en postEvaluada se le quita '0' en código ASCII

e1.d=postEvaluada[i]-'0';

Push(&mi\_pila,e1);

}

//De lo contrario si la postfija evaluada tiene operandos se hace casos para cada operando

else switch(postEvaluada[i]){

//caso que si el operando es "+" hay dos elementos de caracter entero que se van a sumar.

case '+':

e1 = Pop(&mi\_pila);

e2= Pop(&mi\_pila);

res = e2.d + e1.d;

e2.d=res;

Push(&mi\_pila,e2);

break;

//caso que si el operando es "-" hay dos elementos de caracter entero que se van a restar.

case '-' : e1 = Pop(&mi\_pila);

e2= Pop(&mi\_pila);

res = e2.d - e1.d;

e2.d=res;

Push(&mi\_pila,e2);

break;

//caso que si el operando es "\*" hay dos elementos de caracter entero que se van a multiplicar.

case '\*' : e1 = Pop(&mi\_pila);

e2= Pop(&mi\_pila);

res = e2.d \* e1.d;

e2.d=res;

Push(&mi\_pila,e2);

break;

//caso que si el operando es "/" hay dos elementos de caracter entero que se van a dividir.

case '/' : e1 = Pop(&mi\_pila);

e2= Pop(&mi\_pila);

res = e2.d / e1.d;

e2.d=res;

Push(&mi\_pila,e2);

break;

//caso que si el operando es "^" hay dos elementos de caracter entero que se le aplica la potencia de uno.

case '^' : e1 = Pop(&mi\_pila);

e2= Pop(&mi\_pila);

e3.d=1;

//Sumatoria de las multiplicaciones de elemento 3

for(j=0;j<e1.d;j++) e3.d=e3.d\*e2.d;

Push(&mi\_pila,e3);

break;

}

i++;

}

e2=Pop(&mi\_pila);

printf("\n\nEl resultado de la expresion es: %i \n", e2.d);

}

//Si no se ha ingresado nada se manda un mensaje.

else

puts("Aun no se ha evaluado una expresion a posfijo.\n");

system("pause");

system("cls");

return 0;

}

* Bibliografía (en formato IEEE)

[1] Edgardo Adrián Franco Martínez, “Practica 01: Evaluación de expresiones infijas”, Octubre 2015. [En línea]. Disponible en: <http://eafranco.com/docencia/estructurasdedatos/files/practicas/Practica01.pdf>

[2] Edgardo Adrián Franco Martínez, “TADPilaDocumentada.rar”, Octubre 2015. [En línea]. Disponible en:

<http://eafranco.com/docencia/estructurasdedatos/files/practicas/01/TADPilaDocumentada.rar>