Práctica 2

Equipo:

Los tíos pelones

Integrantes:

Buendía Velazco Abel

Carpio Becerra Erick Gustavo

Hernández Molina Leonardo Gaell

Velázquez Diaz Luis Francisco

Grupo:

2CM1

Tabla de Contenidos

Marco Teórico

Las estructuras de datos en el ámbito de la programación son una manera de representar información en una computadora. Cuentan con un comportamiento interno, es decir, estas se rigen por reglas o restricciones específicas que han sido diseñadas con base en el funcionamiento interno de la estructura en cuestión.

Una estructura de datos permite al desarrollador de código organizar la información de manera eficiente. Permite trabajar en un nivel de abstracción alto para almacenar la información y posteriormente acceder a ella, modificarla y, en general, manipularla. Por lo tanto, podemos usarlas para diseñar una solución correcta y eficiente para un determinado problema.

La Pila

Si bien existen muchas otras estructuras de datos, en esta práctica únicamente empleamos la de la pila. Una pila es una estructura de datos de entradas ordenadas tales que solo se introduce y elimina por un extremo al que usualmente se le llama cima o tope.

La pila se encuentra dentro de las estructuras de datos dinámicas ya que se amplia y contrae durante la ejecución del programa, permitiendo que la estructura contenga únicamente nodos (elementos) en uso, haciendo uso eficiente de la memoria; a diferencia de los típicos arreglos, los cuales contienen espacio para almacenar un número fijo de elementos. También es una estructura de datos lineal debido a que los elementos que contiene ocupan lugares sucesivos y cada uno de ellos tienen un único sucesor y predecesor, es decir, sus elementos están ubicados uno al lado del otro relacionados de forma lineal.

La pila es un tipo especial de lista enlazada que permite almacenar y recuperar datos empleando el criterio LIFO (Last In, First Out), lo que significa que el último elemento en entrar a la pila será el primero en salir. Emplea dos operaciones básicas inversas:

Push (apilar), coloca un elemento en la pila y Pop (desapilar), retira el último elemento de la pila.

Expresiones infijas

La notación infija es la manera más común que se utiliza para escribir las expresiones u operaciones matemáticas, es la que utilizamos en la vida cotidiana. En esta notación los operadores se escriben entre los operandos y, por ende, es necesario el uso de paréntesis para determinar el orden en el que se deben de ejecutar las operaciones cuando estas son más complejas.

Expresiones posfijas

En la notación posfija, las expresiones matemáticas se escriben en el orden: primer operando, segundo operando y al último el operador. Como en el caso de la notación

infija, las operaciones son evaluadas de izquierda a derecha, cuando se encuentra un operador, se ha de realizar el cálculo correspondiente con los dos operandos anteriores a este, el resultado se considera como un nuevo operando.

Una característica importante de esta notación es que se elimina por completo el uso de paréntesis, lo que hace más sencillo el análisis de las expresiones por las computadoras.

Planteamiento del problema

La función principal que busca el algoritmo de esta práctica es convertir una expresión matemática en notación infija a posfija empleando la estructura de datos pila para facilitar esta tarea. Para ello, nos dimos cuenta de que es más fácil encontrar la solución si fragmentamos el problema inicial en tres partes que validen respectivamente aspectos de la expresión introducida por el usuario para que de esta manera se reduzcan al máximo las posibilidades de que el programa falle o se rompa. Tenemos entonces que los pasos para pasar una expresión matemática de notación infija a posfija son los siguientes:

1. Evaluación de paréntesis

Este subproblema busca decirnos si los paréntesis, llaves y corchetes están bien colocados en la expresión. Básicamente buscamos saber si todos los paréntesis, llaves y corchetes abren y cierran en orden, y que siempre se apertura un paréntesis, exista un paréntesis de cierre.

2. Conversión a posfijo

Ahora que ya validamos los paréntesis, sabemos que la expresión debe estar parcialmente correcta, ahora es tarea de la conversión infija-posfija determinar la precedencia de los operadores para efectuar la conversión.

Otro punto que considerar en este subproblema es que el algoritmo evalúe si la posición de los operadores es correcta, de otro modo, se obtendrá un error de ejecución debido a que el usuario estaría introduciendo de manera errónea la expresión matemática.

3. Evaluación de la expresión posfija

Ya con la expresión equivalente en posfijo, el programa buscará obtener por medio del usuario los valores de todas las variables que haya utilizado en la expresión matemática introducida. Entonces evaluará la expresión posfija con estos valores para finalmente obtener un valor como resultado de efectuar la operación.

Diseño y funcionamiento de la solución

1. Evaluación de paréntesis

Este resulta ser una parte sencilla de la solución del problema cuando empleamos la pila, ya que cabe recalcar que, sin su ayuda, este problema sería más complicado de abstraer. Simplemente bastará con recorrer toda la cadena que el usuario introdujo por la entrada estándar en busca de paréntesis de apertura, efectuando lo siguiente:

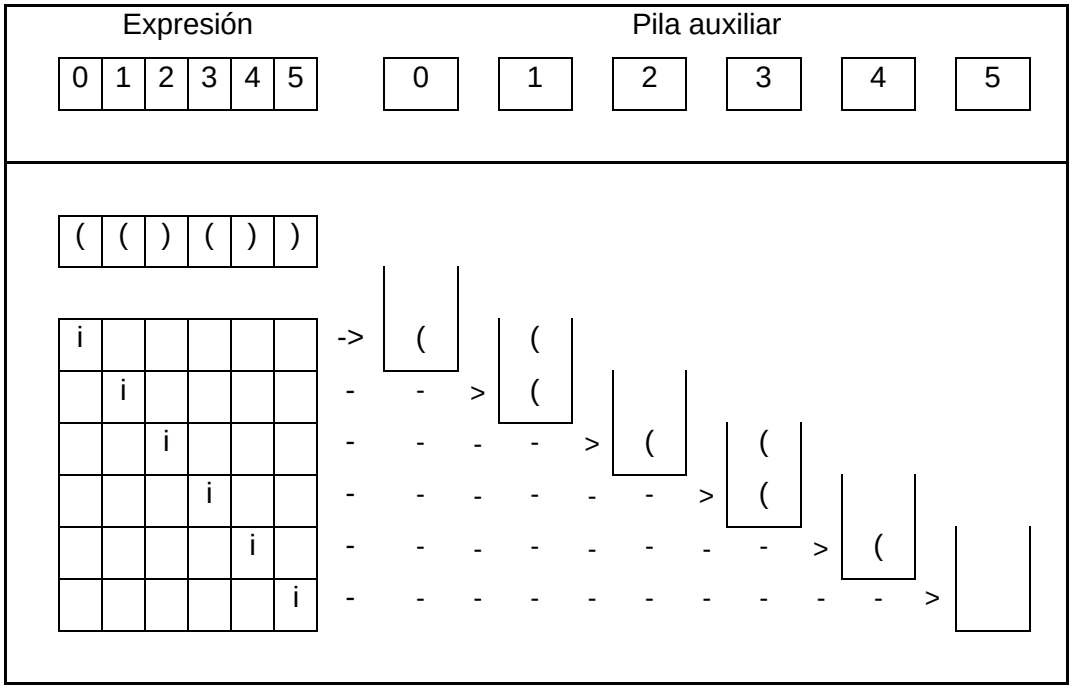
Si un paréntesis abre, se mete a la pila; si un paréntesis cierra, se hace pop a la pila auxiliar.

De este modo se generan tres posibles resultados con sólo uno favorable:

Caso 1 (expresión válida)

La pila al final de efectuar la validación está vacía.

Este caso puede verse mejor en la siguiente figura:

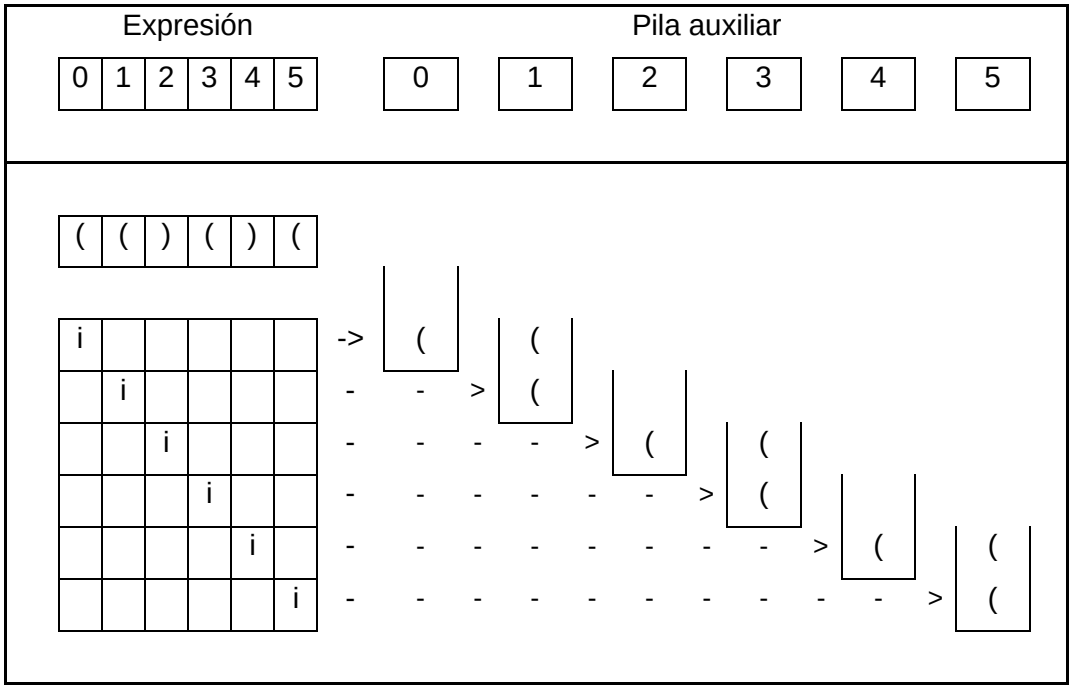


Como vemos, en la posición final (i=5), la pila auxiliar está vacía.

Caso 2 (expresión inválida)

La pila al final de efectuar la validación no está vacía.

Este caso puede verse mejor en la siguiente figura:

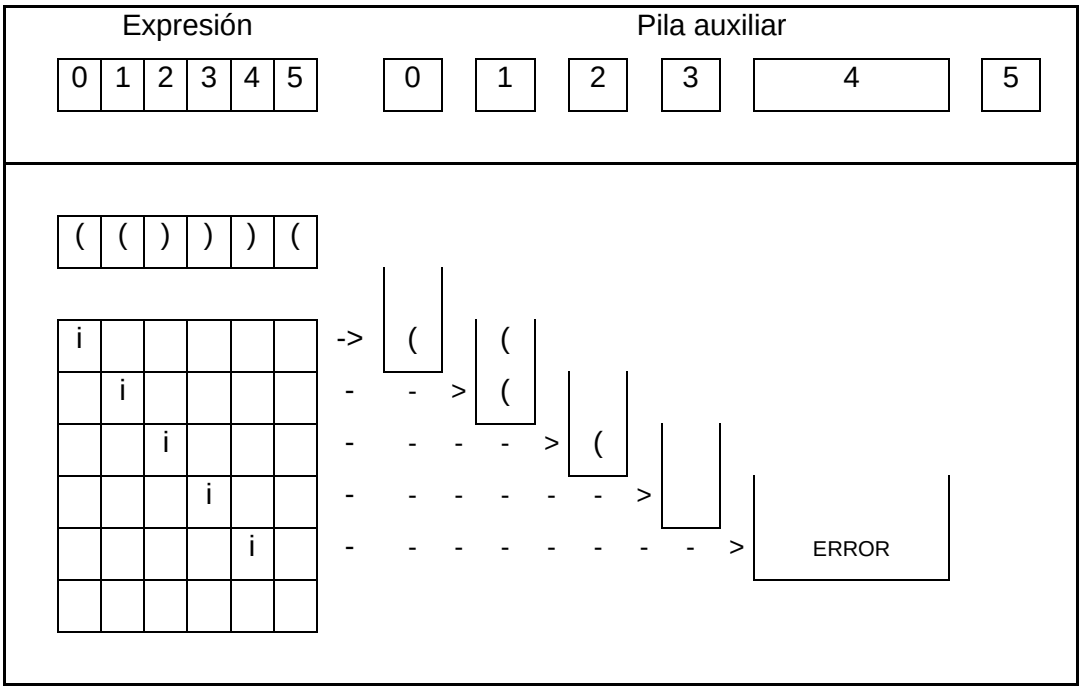


Como vemos, en la posición final (i=5), la pila auxiliar contiene dos elementos, por lo que no está vacía.

Caso 3(expresión inválida)

Durante la ejecución del algoritmo se hace un pop a la pila vacía (subdesbordamiento de pila, por definición en el TAD pila).

Este caso puede verse mejor en la siguiente figura:



Como vemos, ni siquiera puede llegar a la posición final del arreglo (i=5), ya que en i=4 se generó un error por subdesbordamiento de pila.

Paréntesis, corchetes y llaves

Para incorporar los tres tipos de agrupadores, aprovechando que la función pop retorna el valor que se saca de la pila, entonces bastará con incluir los caracteres ‘[’ ‘{’ en la búsqueda junto con el ‘(’, y en caso de encontrar ‘)’, ‘]’ o ‘}’, si el valor que retorna pop es diferente a ‘(’ , ‘[’ o ‘{’ respectivamente, entonces invalidar la cadena.

2. Conversión a posfijo

Cuando hablamos de la conversión a notación posfija, puede que nos estemos refiriendo a la parte más laboriosa del programa, puesto a que implica bastantes condicionales para su correcta ejecución. Por ello, nosotros nos apoyamos de tres funciones (una principal y 2 complementarias) que nos permiten dividir el subproblema y de igual forma auto documentan el código, permitiendo una lectura más clara y ordenada al desarrollador.

Las funciones diseñadas fueron:

EsOperando

Esta resulta ser la función más básica, pues únicamente pide como parámetro un elemento y devuelve 1 si el elemento se encuentra entre la a y la z o la A y la Z, y 0 en caso contrario.

Precedencia de los operadores

Esta función asigna un valor de prioridad a los operadores siguiendo la jerarquía de operaciones. Recibe como argumento un elemento y esta debe devolver su valor de prioridad en función al operador en cuestión o -1 en caso de que no sea ninguno de los operadores especificados.

Los valores de prioridad asignados por la función son los siguientes:

- Suma o resta = 1
- Multiplicación o división = 2
- Potencia = 3

Entonces, tenemos que la función evalúa el elemento por medio de un switch en donde los casos varían dependiendo del elemento, ya sea si encuentra '+', '-','*', '/' o '^' y devuelve su respectivo valor de prioridad.

Infija-a-Posfija

Con ayuda de las funciones anteriores, ya no resulta tan complicado implementar esta función, pues entonces, se hace lo siguiente:

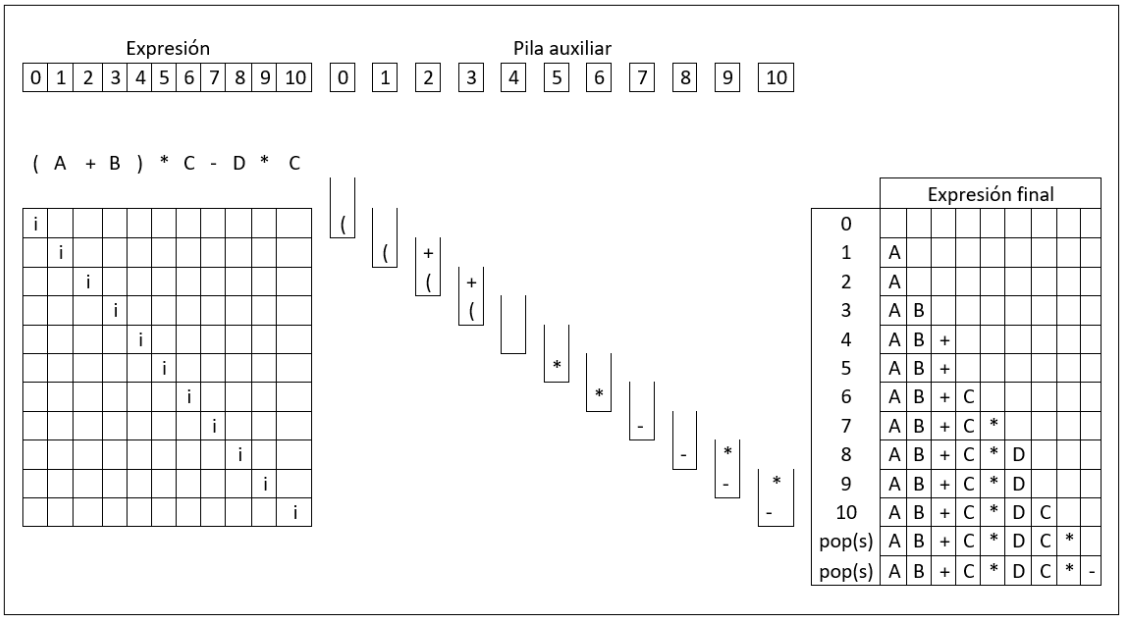
La función recibirá como argumento la expresión introducida por el usuario y ya validada por la función ValidaParentesis, y devolverá -1 en caso de que el elemento no sea un operador (que sea un operando).

La pila se va recorriendo de izquierda a derecha, si se encuentra una variable, esta deberá ser puesta inmediatamente en el arreglo que al finalizar la función deberá contener la expresión en notación postfija. De otro modo, si el elemento corresponde a un paréntesis de apertura '(', este deberá introducirse en una pila auxiliar. Si el elemento es un paréntesis que cierra ')', se deberá hacer pop a la pila e ir guardando esos elementos salientes en la cadena final hasta que el elemento que salga sea igual al paréntesis de apertura ')', siempre, claro, que la pila no esté vacía. Si lo anterior se incumple, mientras que la precedencia del

elemento de la expresión sea menor o igual a la del tope de la pila, se deberá hacer pop a la pila e ir guardando los elementos que salgan en la cadena final hasta que el tope de la pila tenga una precedencia menor a la del operador en turno, es entonces cuando se introduce dicho operador en la pila.

Por último, una vez que se haya recorrido toda la expresión, pero la pila no está vacía, se deberá hacer pop y guardar los elementos que salgan de la pila en la cadena final. Como resultado idóneo, habremos obtenido la expresión convertida a notación posfija.

El algoritmo de infija a postfija se puede observar mejor en la siguiente figura:



Podemos observar que el recorrido del arreglo acaba, pero la pila sigue vacía, entonces lo único que queda por hacer es hacer pop a la pila hasta que vacíe completamente.

3. Evaluación de la expresión posfija

Una vez realizada la conversión de notaciones, únicamente queda evaluarla con valores numéricos para obtener un único valor como resultado. Para esto, se le pide al usuario que introduzca manualmente por la entrada estándar los valores de las variables que haya ocupado. Entonces se deberá intercambiar el caracter de la variable por el entero o flotante para evaluar la expresión en posfijo con estos nuevos valores.

La segunda parte de esta función deberá auxiliarse de una pila para realizar de forma más sencilla las operaciones, como la expresión para este punto ya debe estar en notación posfija y con valores numéricos, lo único que queda por hacer es recorrer toda la cadena (de izquierda a derecha), si se encuentra un número, meterlo dentro de la pila auxiliar, de lo contrario, sacar los últimos dos elementos de la pila y sumarlos en caso de que el operador encontrado sea '+', restarlos en caso de que sea '-', multiplicarlos en caso de que sea '*', dividirlos en caso de que sea '/', y elevar el segundo al primero en caso de que sea '^'; para posteriormente meter el resultado de dicha operación a la pila. Para cuando se recorra toda la cadena completa, la pila solo deberá contener un solo elemento (el resultado de toda la evaluación postfija), entonces ya solo hará falta devolver ese elemento y habremos terminado.

Implementación de la solución

Librerías

Las librerías utilizadas son:

- stdio.h
- stdlib.h
- string.h

  Para facilitar la manipulación de cadenas
- ctype.h

  Para identificar los tipos de caracteres
- pila_din.h (TAD pila)

  El tipo de dato abstracto con el que opera nuestra estructura de datos pila, necesario para implementar todos los algoritmos de la práctica.

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<string.h>
4   #include <ctype.h>
5   #include "C:\Users\Gustavo Carpio\Desktop\Practica2\pila_din\pila_din.c"
```

Prototipos de las funciones

Las funciones son seis, de las cuales, sólo las tres principales son las que giran en torno al TAD pila (encontrado en los archivos provistos en el apartado de la práctica en la plataforma): VerificarParentesis, infixToPostfix y EvalPosfija.

```
// Prototipos de las funciones
int isOperand(char);
int VerificarParentesis(char[]);
void cargarFormula(char[]);
int Prec(char);
int infixToPostfix(elemento[]);
int EvalPosfija(elemento *);
```

Main

La implementación de funciones en la práctica nos permite tener un main muy limpio, pues en este sólo basta con declarar la cadena en la que el usuario introducirá la expresión en infijo, declarar el arreglo de valores para la evaluación postfija, llamar a la función que permite al usuario introducir la expresión, validar los paréntesis de la expresión; en caso de que sea válida, llamar a la función de conversión de notaciones y por último llamar a la función que evalúa la expresión postfija, la cual por si misma se encarga de imprimir el resultado de la evaluación.

```
int main()
{
    char e[100];  //Declaraci□n de la cadena e con tama□o 100

    // Llamamos a la funci□n que permite digitar la expresi□n
    cargarFormula(e);

    //Condicion que nos ayuda a evaluar si la expresi□n es correcta
    if (VerificarParentesis(e)){
        puts("Los parentesis estan en orden\n");  //En caso de que sea correcta
        infixToPostfix(e);                        //Convertir la expresion infija a postfija
        EvalPosfija(e);                           //Evaluar la expresion postfija
    }
    else
        puts("Los parentesis estan mal colocados\n");//En caso de que sea incorrecta
    return 0;
```

Funciones

isOperand

Esta función tiene su mayor contribución en la función infixToPostfix que veremos más adelante, ya que nos devuelve un valor de 1 si el elemento que se le entrega como argumento es un operando (una variable) y un 0 si no lo es.

```c
/*
    Son iguales (isOperand): recibe <- char; retorna -> int;
    isOperand(ch)
    Efecto: Si la letra esta en mayuscula o miniscula le da la misma
    precedencia
*/

int isOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') ||
           (ch >= 'A' && ch <= 'Z');
}
```

VerificarParentesis

Se crea e inicializa una pila para que por medio de un ciclo for se recorra toda la cadena que se le manda como argumento, en caso de encontrar cualquier agrupador que abra, este se mete a la pila y se sigue recorriendo el arreglo, en caso de encontrar cualquier agrupador que cierre, se deberá sacar un elemento de la pila, pero este debe corresponder al agrupador que se lee de la cadena al momento de hacer pop, de lo contrario la función devuelve 0, indicando que la expresión fue mal introducida y por lo tanto, los paréntesis no están correctamente colocados.

```c
int VerificarParentesis (char e[])
{
    //Declaraci□n de nuestra pila s
    pila s;

    // Inicializamos nuestra pila s
    Initialize(&s);

    int i; //Declaraci□n de iterador

    for( i=0 ;i < strlen(e) ;i++ ){ //Lee la cadena caracter por caracter
        if (e[i] == '(' || e[i] == '[' || e[i] == '{' ) // Busca si hay algun '(', '{' o '['
        {Push (&s, e[i]);} // Si cumple con la condici□n lo mete a la pila

        else{
        if (e[i] == ')' ){  //Si encuentra un ')'
        if (Pop(&s) != '(')  //Saca de la pila un ')'
        return 0;
         }

        else if (e[i] == ']'){ //Si encuentra un ']'
        if (Pop(&s) != '[')     //Saca de la pila un ']'
        return 0;
        }

        else if (e[i] == '}'){ //Si encuentra un '}'
        if (Pop(&s) != '{')      //Saca de la pila un '}'
        return 0;
        }
            }
                }

        if (Empty(&s))     // Si la pila esta vacia
            return 1;     // Termina de forma correcta
        else
            return 0;     //Termina de forma incorrecta
    }
```

cargarFormula

Es una función void que recibe el arreglo e destinado para la obtención y manipulación de la fórmula matemática, por lo que para cuando se llame esta función, el arreglo deberá estar vacío o contener basura, es entonces que por medio de la entrada estándar (gets) lo llena.

```
53  /*
54      Cargar Formula (cargarFormula): recibe <- char *(formula);
55      cargarFormula(formula)
56      Efecto: Funci□n para digitar la expresion y guardarla en una cadena "formula"
57  */
58
59  void cargarFormula(char *formula)
60  {
61      printf("Ingrese la formula:");
62      fflush(stdin); // Limpia la entrada de el bufer
63      gets(formula); // Guarda la cadena introducida en "formula"
64      return;
65  }
```

Prec

Esta función entera recibe un caracter y devuelve el orden de precedencia del operador que se haya mandado como argumento. Por medio de un switch evalúa por casos: si el carácter es ‘+’ o ’-’, retorna 1; si es ’*’ o ’/’, retorna 2; y si es ’^’, retorna 3.

Esta función es usada en la función principal infixToPostfix para llevar a cabo correctamente la jerarquía de operaciones.

```
111 /*
112     Precedencia (Prec): recibe <- char; return -> int;
113     Prec(ch)
114     Efecto: Asigna un valor de prioridad dependiendo de el caracter
115 */
116
117 int Prec(char ch)
118 {
119     switch (ch){
120         case '+':
121         case '-':
122             return 1; //Verifica el operador y le da una prioridad de 1
123
124         case '*':
125         case '/':
126             return 2; //Verifica el operador y le da una prioridad de 2
127
128         case '^':
129             return 3; //Verifica el operador y le da una prioridad de 3
130     }
131     return -1;
132 }
```

infixToPostfix

Esta función entera, recibe un apuntador al inicio del arreglo e y devuelve -1 en caso de que algo haya salido mal durante la conversión y 0 en caso de que haya concluido la conversión con éxito. Esta función modifica el mismo arreglo de elementos (el arreglo e) conforme se va ejecutando, esto se logra con la ayuda de 2 iteradores (i=0 , k=-1), los cuales se encuentran desfasados con una posición de diferencia, por lo que el iterador k sobrescribe en la posición que el iterador i ya evaluó; de esta manera se va llenando nuevamente el arreglo y no tiene por que generar algún problema en la ejecución. Una vez que realizó el proceso hasta la línea 169, en el penúltimo paso (antes de imprimir la expresión en pantalla) hace que en el pre incremento de la posición k guarde el caracter reservado '\0', lo que indica el fin de cadena y por lo tanto, desecha el resto de caracteres que podría todavía contener la cadena que se está sobrescribiendo.

```
127 /*
128     Infija a postfija (infixToPostfix): recibe <- elemento *(expresion);
129     infixToPostfix(expresion)
130     Efecto: Cambia una expresi□n infija a una expresi□n postfija
131 */
132
133 int infixToPostfix(elemento *expresion)
134 {
135     int i, k; //Declaraci□n de iteradores
136     pila stack; //Declaraci□n de pila
137     Initialize(&stack); //Inicializaci□n de pila
138
139     for (i = 0, k = -1; expresion[i]; ++i){
140
141         if (isOperand(expresion[i])) //Verifica si el caracter es mayuscula o miniscula
142             expresion[++k] = expresion[i]; // En ls posici□n 0(++k) guardas lo que tienes en la posici□n 1 (i++)
143
144         else if (expresion[i] == '(') // Si el caracter de la posici□n actual es = '('
145             Push(&stack, expresion[i]); // Mete a la pila el caracter
146
147         else if (expresion[i] == ')'){ // Si el caracter de la posici□n actual es = ')'
148
149
150             while (!Empty(&stack) && Top(&stack) != '(') // Mientras la pila no este vacia y el tope no sea '('
151                 expresion[++k] = Pop(&stack); //En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)
152
153             if (!Empty(&stack) && Top(&stack) != '(') //Si la pila no este vacia y el tope no sea '(' (despues de el ciclo)
154                 return -1; //El programa termina de manera incorrecta
155
156             else // En caso de que no se cumpla
157                 Pop(&stack); // Saca el caracter de la pila
158         }
```

```
159         else {
160             while (!Empty(&stack) &&                    //Mientras la pila no sea vacia
161                 Prec(expresion[i]) <= Prec(Top(&stack)))   // y la precedencia de el operador actual sea menor a la precedencia del tope
162                 expresion[++k] = Pop(&stack);    // En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)
163             Push(&stack, expresion[i]);          // Metemos el caracter a la pila
164         }
165     }
166
167     while (!Empty(&stack))      //Mientras la pila no sea vacia
168         expresion[++k] = Pop(&stack ); //En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)
169
170     expresion[++k] = '\0';      //Guarda el final la cadena utilizando el caracter nulo '\'
171     printf( "\nEl resultado de la expresion es: \n %s", expresion);
172     return 0;
173 }
```

EvalPosfija

Esta función está planeada para recibir el arreglo de elementos e (que contiene la expresión ya en notación posfija y con valores numéricos) y devolver el último elemento de la pila (el cual debe ser el resultado de la evaluación postfija posterior a la ejecución del algoritmo de evaluación).

```
181  int EvalPosfija(elemento *exp){
182
183      int i; //Declaraci□n iterador
184
185      pila pilita; //Declaraci□n pila
186
187      Initialize(&pilita); //Inicializaci□n de la pila
188
189      for (i = 0; exp[i]; ++i){
190
191          if (isdigit(exp[i]))        //Si el caracter que se esta leyendo es un digito, lo mete a la pila
192              Push(&pilita, exp[i] - '0');
193
194          else{
195              int val1 = Pop(&pilita); //Guarda el ultimo digito de pila
196              int val2 = Pop(&pilita);
197
198              switch (exp[i]){ //Evalua el operador
199
200              case '+': Push(&pilita, val2 + val1); //En caso que el operador sea '+', se suman los 2 digitos guardados
201                  break;
202              case '-': Push(&pilita, val2 - val1); //En caso que el operador sea '-', se restan los 2 digitos guardados
203                  break;
204              case '*': Push(&pilita, val2 * val1); //En caso que el operador sea '*', se multiplican los 2 digitos guardados
205                  break;
206              case '/': Push(&pilita, val2/val1); //En caso que el operador sea '/', se dividen los 2 digitos guardados
207                  break;
208              }
209          }
210      }
211
212      return Pop(&pilita); //Retorna el ultimo elemento de la pila
213  }
```

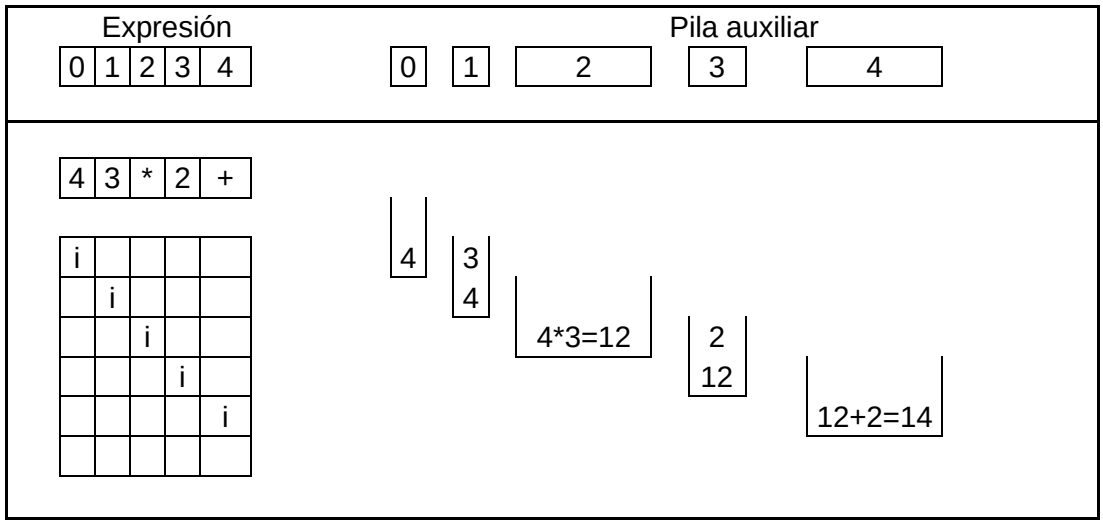
Evaluación de la expresión posfija

Una vez realizada la conversión de notaciones, únicamente queda evaluarla con valores numéricos para obtener un único valor como resultado. Para esto, se le pide al usuario que introduzca manualmente por la entrada estándar los valores de las variables que haya ocupado. Entonces se deberá intercambiar el carácter de la variable por el entero o flotante para evaluar la expresión en posfijo con estos nuevos valores.

La segunda parte de esta función deberá auxiliarse de una pila para realizar de forma más sencilla las operaciones, como la expresión para este punto ya debe estar en notación posfija y con valores numéricos, lo único que queda por hacer es recorrer toda la cadena (de izquierda a derecha), si se encuentra un número, meterlo dentro de la pila auxiliar, de lo contrario, sacar los últimos dos elementos de la pila y sumarlos en caso de que el operador encontrado sea ‘+’, restarlos en caso de que sea ‘-’, multiplicarlos en caso de que sea ‘*’, dividirlos en caso de que sea ‘/’, y elevar el segundo al primero en caso de que sea ‘^’; para

posteriormente meter el resultado de dicha operación a la pila. Para cuando se recorra toda la cadena completa, la pila solo deberá contener un solo elemento (el resultado de toda la evaluación postfija), entonces ya solo hará falta devolver ese elemento y habremos terminado.

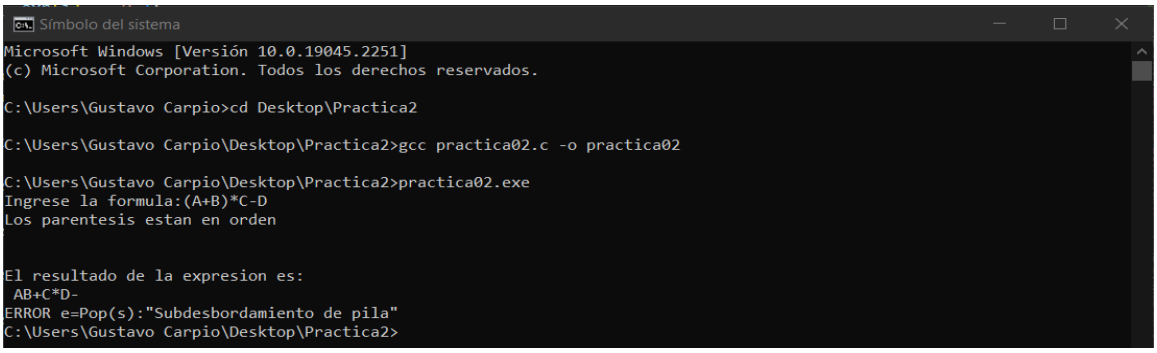
Se observa mejor el proceso de evaluación en la siguiente figura:



Como vemos, al final, la pila tiene el resultado de la expresión evaluada, por lo que solo hace falta sacarlo y mostrarlo en pantalla.

Funcionamiento

Prueba 1



Prueba 2

Errores detectados

Desafortunadamente, en esta práctica tuvimos errores bastante puntuales, cuando realizamos el análisis para su resolución, lo entendimos de una manera errónea.

Comenzando con que entendimos que la operación que colocábamos en cada uno de los rubros debía ser diferente para cada aspecto, cuando una operación debía guardarse y utilizarse en cada una de las opciones.

Así mismo, creímos que implementando un menú el programa final podía ser más intuitivo para el usuario, pero al parecer no fue la mejor opción, aunque esto le daba mucha estética y le daba un ambiente agradable, nos concentramos más en eso que en lograr el funcionamiento óptimo de la práctica, eso nos hace darnos cuenta de que “menos es más”.

Otro de los problemas que tuvimos fue que en la parte donde se le debía asignar valores a las letras para poder evaluar la expresión, nuestro programa se “rompía” y se cerraba completamente si intentábamos darle un valor al carácter. Nuestra práctica si lograba dar el resultado de las expresiones solo si usábamos valores numéricos, pero con letras el resultado siempre nos daba cero.

Y para concluir con nuestros errores, pensamos que debíamos implementar ambas pilas en el programa (tanto la estática y la dinámica) y al momento de presentarlo, nos dimos cuenta de que usamos las mismas variables para ambas pilas, lo que hace que no tuviese ningún sentido lo que elaboramos para usar ambas stacks, entonces en general nuestra propuesta para llegar a la solución dejó bastante que desear.

```
void menu_vis (){
  printf ("BIENVENIDO AL MENU DE SELECCION\n");

  puts("-----------------------------------------------\n");

  printf ("Opcion 1: Verificar parentesis estatico\n");
  printf ("Opcion 2: Verificar parentesis dinamico\n");

  puts("-----------------------------------------------\n");

  printf ("Opcion 3: Expresion infija a postfija estatico\n");
  printf ("Opcion 4: Expresion infija a postfija dinamico\n");

  puts("-----------------------------------------------\n");

  printf ("Opcion 5: Evaluar expresion postfija estatico\n");
  printf ("Opcion 6: Evaluar expresion postfija dinamico\n");

  puts("-----------------------------------------------\n");

  printf ("Opcion 7: Salir del programa\n");

  puts("");
```

```
for (i = 0; exp[i]; ++i){

    if (isdigit(exp[i]))        //Si el caracter que se esta leyendo es un digito, lo transforma a un caracter
        Push(&pilita, exp[i] - '0'); //Lo mete a la pila

    else{
        int val1 = Pop(&pilita); //Guarda el ultimo digito de pila
        int val2 = Pop(&pilita);

        switch (exp[i]){ //Evalua el operador

        case '+': Push(&pilita, val2 + val1); //En caso que el operador sea '+', se suman los 2 digitos guardados
            break;
        case '-': Push(&pilita, val2 - val1); //En caso que el operador sea '-', se restan los 2 digitos guardados
            break;
        case '*': Push(&pilita, val2 * val1); //En caso que el operador sea '*', se multiplican los 2 digitos guardados
            break;
        case '/': Push(&pilita, val2/val1); //En caso que el operador sea '/', se dividen los 2 digitos guardados
            break;
        }
    }
}
```

Ilustración 1: Código donde no logramos ponerle valor a las letras

```
#include "C:\Users\luisv\OneDrive\Escritorio\Practica2\Ejercicio1-Validar_Parentesis\Estatico\Validar_Est.c"
#include "C:\Users\luisv\OneDrive\Escritorio\Practica2\Ejercicio1-Validar_Parentesis\Dinamico\Validar_Din.c"

#include "C:\Users\luisv\OneDrive\Escritorio\Practica2\Ejercicio2-InfijaPostfija\Dinamico\InfijaPostfija_Din.c"
#include "C:\Users\luisv\OneDrive\Escritorio\Practica2\Ejercicio2-InfijaPostfija\Estatico\InfijaPostfija_Est.c"

#include "C:\Users\luisv\OneDrive\Escritorio\Practica2\Ejercicio3-EvaluarPostfija\Pila_Dinamica\EvaluaPost_Din.c"
#include "C:\Users\luisv\OneDrive\Escritorio\Practica2\Ejercicio3-EvaluarPostfija\Pila_Estatica\EvaluaPost_Est.c"
```

Ilustración 3: Uso de ambas pilas al mismo tiempo

Posibles mejoras

Primero debemos encontrar la manera de lograr que la expresión se coloque al intentar validar los paréntesis se guarde y termine siendo la misma que pase de infija a posfija la evalué de igual manera, posterior a ello, si queremos ir por la misma línea de nuestra solución, cambiar las variables del “.h” para alguna de nuestras pilas para que el programa pueda identificar una de otra, también, se tiene que adaptar para que si el usuario te coloca letras, despliegue un apartado donde el mismo deba darle valor a esas letras y nos de el valor correcto sin que este colapse.

Y por último verificar si el menú es necesario y de no ser así retirarlo para evitar tener exceso de código y hacer más confuso el mismo.

Ilustración 3: Código de Infijo a posfijo mejorado

```
int infixToPostfix(elemento *expresion)
{
    int i, k; //Declaraci n de iteradores
    pila stack; //Declaraci n de pila
    Initialize(&stack); //Inicializaci n de pila

    for (i = 0, k = -1; expresion[i]; ++i){

        if (isOperand(expresion[i])) //Verifica si el caracter es mayuscula o miniscula
            expresion[++k] = expresion[i]; // En ls posici n 0(++k) guardas lo que tienes en la posici n 1 (i++)

        else if (expresion[i] == '(') // Si el caracter de la posici n actual es = '('
            Push(&stack, expresion[i]); // Mete a la pila el caracter

        else if (expresion[i] == ')'){ // Si el caracter de la posici n actual es = ')'


            while (!Empty(&stack) && Top(&stack) != '(') // Mientras la pila no este vacia y el tope no sea '('
                expresion[++k] = Pop(&stack); //En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)

            if (!Empty(&stack) && Top(&stack) != '(') //Si la pila no este vacia y el tope no sea '(' (despues de el ciclo)
                return -1; //El programa termina de manera incorrecta

            else // En caso de que no se cumpla
                Pop(&stack); // Saca el caracter de la pila
        }
        else {
            while (!Empty(&stack) &&                        //Mientras la pila no sea vacia
                Prec(expresion[i]) <= Prec(Top(&stack)))    // y la precedencia de el operador actual sea menor a la precedencia del tope
                expresion[++k] = Pop(&stack);     // En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)
            Push(&stack, expresion[i]);           // Metemos el caracter a la pila
        }
    }

    while (!Empty(&stack))      //Mientras la pila no sea vacia
        expresion[++k] = Pop(&stack ); //En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)

    expresion[++k] = '\0';      //Guarda el final la cadena utilizando el caracter nulo '\'
    printf( "\nEl resultado de la expresion es: \n %s", expresion);
    return 0;
}
```

Ilustración 4:Evaluacion de posfijo mejorado, pero aún sin lograr darle valor a las letras

Conclusiones

Conclusión Buendía Velazco Abel:

En general me pareció una práctica muy buena para entender como es el funcionamiento de la estructura de datos “pila”, quedo muy claro cómo es se hizo la implementación de su TAD, al igual que su funcionamiento, lo que al equipo se le dificulto fue la hacer correctamente el tercer ejercicio, ya que al momento de asignar valores a los caracteres que se introducían en el buffer de la consola, no funcionaban correctamente y por lo tanto decidimos no incluirlo en el programa final, al igual que por mala interpretación del equipo el programa final no se presentó como uno solo que ejecutaba los 3 programas en uno solo, sino que se agregó un menú para hacer cada programa por separado, pero son errores que en las siguientes practicas iremos mejorando y de igual manera tratar de no cometer errores similares, incluso tratar de no tener ningún error, pero como se mencionó anteriormente esta práctica nos nuestra de manera sencilla de cómo es que funciona una pila, además que en la investigación del marco teórico se encontraron más aplicaciones, como lo puede ser el almacenamiento de variables e instrucciones constituidas por métodos y después ser ejecutados mediante una máquina virtual, quedo muy claro que es una pila y cómo funciona, espero que en las próximas prácticas de las estructuras de datos sean así y no quedar con ninguna duda ni error que corregir.

Carpio Becerra Erick Gustavo

En esta práctica pude darme cuenta de la importancia de los distintos tipos de notaciones en las que se escriben las expresiones matemáticas, ya que la jerarquía de operaciones en notación infija puede que no sea tan trivial para la computadora, por el contrario, en notación postfija y con ayuda de una pila auxiliar, en realidad resulta bastante sencillo y simple. No obstante, en la vida cotidiana no se suele usar la notación postfija, por ello puede resultar un reto diseñar la solución para un programa que haga el cambio de notaciones en expresiones, sin embargo, con el uso de la estructura de datos pila bien implementada, se puede abstraer mucho más fácil la solución para la conversión de notaciones (infija a postfija en este caso).

Es ahora un poco más tangible para mí el cuan relevantes son las estructuras de datos cuando hablamos de programación. Sin duda esta práctica hubiera sido mucho más difícil sin el uso de la pila y me pone a pensar qué otros usos importantes tiene.

Conclusión Hernández Molina:

Elaborar esta práctica fue una tarea que parecía nada complicada, pero al ir desarrollando el proyecto nos fuimos encontrando con ciertas disyuntivas las cuales nos hacían preguntarnos que sería mejor utilizar, es decir, en qué casos nos convenia usar un while o un for para ciertas cosas. Además, si bien, ya tenía buenos conocimientos acerca de la popularización, en verdad no había tenido una buena oportunidad de aplicarla muy bien y eso precisamente es lo que hace poder realizar la práctica de una manera sencilla. Ya que podemos evitarnos muchas complicaciones y muchas redundancias en nuestro código si simplemente modularizamos en partes nuestro programa podemos reutilizar muchas cosas en otras funciones.

Por otro lado, la práctica me ayudó mucho a reforzar otros conocimientos adquiridos durante las otras materias cursadas, así como aprendí algunas otras cosas nuevas, entre ellas la librería de string.h la cual si bien utilizamos solo de forma somera, nos dimos a la tarea de investigar un poco más allá de lo que usamos y nos dimos cuenta gracias a ello que es muy útil al momento de estar trabajando con operaciones cómo estas, ya que gracias a ella podíamos saber la posición, copiar la cadena, obtener el tamaño entre muchas otras cosas las cuales fueron de gran ayuda para la realización de la práctica.

Conclusión Velázquez Díaz Luis Francisco:

En lo particular creí que sería una práctica más sencilla de lo que fue, nos tomó mucho tiempo y al final no logramos los resultados esperados, pero gracias a ésta, pude darme cuenta de que ciertos temas de programación que utilice anteriormente que creía que no iban a tener un uso en un futuro, eran muy útiles para realizar esta práctica y pudieron ahorrarnos tiempo y código de a verlos implementado.

A su vez, logré entender que las pilas son muy útiles para resolver muchos tipos de problemas, cada trabajo que realizamos sobre estructuras de datos, me doy cuenta de que su uso te ahorra demasiadas cuestiones.

Por otro lado, me ayudó a reforzar muchos de los aprendizajes que obtuvimos a lo largo de la semana, también, el cometer muchos errores en esta práctica, tener un entendimiento muy malo de la misma, nos va a ayudar a leer y comprender los siguientes ejercicios de mejor manera y así tener mejores resultados. De los errores se aprende y este es un buen momento para aprender y mejorar.

ANEXOS

Evaluación Posfija

```
/*
EvaluaPost_Din.c
V 1.0 Noviembre 2022
-----------------------------------------------------
Autores: Los tíos pelones
        Grupo: 2CM1
        Buendia Velazco Abel
        Carpio Becerra Erik Gustavo
        Hernandez Molina Leonardo Gaell
        Velazquez Diaz Luis Francisco
-----------------------------------------------------
Programa que evalua (calcula) el valor de una expresión postfija utilizando una pila estatica.
-----------------------------------------------------

Compilacion:
Windows: gcc EvaluaPost_Din.c -o EvaluaPost_Din.exe
Windows: gcc EvaluaPost_Din.c -o EvaluaPost_Din
*/

// Librerias a usar para la ejecucion del programa
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <ctype.h>


// Prototipos de las funciones
int EvalPosfija(elemento * exp);
void cargarFormula(char *formula);


/*
    Cargar Formula (cargarFormula): recibe <- char *(formula);
    cargarFormula(formula)
    Efecto: Función para digitar la expresion y guardarla en una cadena "formula"
*/


/*
    Evalua Postfija (EvalPosfija): recibe <- elemento *(exp); devuelve -> int
    EvalPosfija(exp)
    Efecto: Calcula el valor de una expresión postfija
*/
int EvalPosfija(elemento *exp){

    pila pilita; //Declaración pila
    int i; //Declaración iterador

    Initialize(&pilita); //Inicialización de la pila

    for (i = 0; exp[i]; ++i){

        if (isdigit(exp[i]))        //Si el caracter que se esta leyendo es un digito, lo transforma a un caracter
            Push(&pilita, exp[i] - '0'); //Lo mete a la pila

        else{
            int val1 = Pop(&pilita); //Guarda el ultimo digito de pila
            int val2 = Pop(&pilita);

            switch (exp[i]){ //Evalua el operador

            case '+': Push(&pilita, val2 + val1); //En caso que el operador sea '+', se suman los 2 digitos guardados
                break;
            case '-': Push(&pilita, val2 - val1); //En caso que el operador sea '-', se restan los 2 digitos guardados
                break;
            case '*': Push(&pilita, val2 * val1); //En caso que el operador sea '*', se multiplican los 2 digitos guardados
                break;
            case '/': Push(&pilita, val2/val1); //En caso que el operador sea '/', se dividen los 2 digitos guardados
                break;
            }
        }
    }

    return Pop(&pilita); //Retorna el ultimo elemento de la pila
```

Infija a posfija

```
InfijaPostfija_Din.c
V 1.0 Noviembre 2022
-----------------------------------------------------
Autores: Los tíos pelones
        Grupo: 2CM1
        Buendia Velazco Abel
        Carpio Becerra Erik Gustavo
        Hernandez Molina Leonardo Gaell
        Velazquez Diaz Luis Francisco
-----------------------------------------------------
Programa que convierte una expresión infija a una expresión postfija, utilizando una pila dinamica.
-----------------------------------------------------

Compilacion:
Windows: gcc InfijaPostfija_Din.c -o InfijaPostfija_Din.exe
Windows: gcc InfijaPostfija_Din.c -o InfijaPostfija_Din
*/

// Librerias a usar para la ejecucion del programa
#include<stdio.h>
#include<stdlib.h>
#include<string.h>


// Prototipos de las funciones
void cargarFormula(char *formula);
int isOperand(char ch);
int Prec(char ch);
int infixToPostfix(elemento *expresion);

int isOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') ||
           (ch >= 'A' && ch <= 'Z');
}


int Prec(char ch)
{
    switch (ch){
        case '+':
        case '-':
            return 1; //Verifica el operador y le da una prioridad de 1

        case '*':
        case '/':
            return 2; //Verifica el operador y le da una prioridad de 2

        case '^':
            return 3; //Verifica el operador y le da una prioridad de 3
    }
    return -1;
}

/*
    Infija a postfija (infixToPostfix): recibe <- elemento *(expresion);
    infixToPostfix(expresion)
    Efecto: Cambia una expresión infija a una expresión postfija
*/

int infixToPostfix(elemento *expresion)
{
    int i, k; //Declaración de iteradores
    pila stack; //Declaración de pila
    Initialize(&stack); //Inicialización de pila

    for (i = 0, k = -1; expresion[i]; ++i){

        if (isOperand(expresion[i])) //Verifica si el caracter es mayuscula o miniscula
            expresion[++k] = expresion[i]; // En ls posición 0(++k) guardas lo que tienes en la posición 1 (i++)

        else if (expresion[i] == '(') // Si el caracter de la posición actual es = '('
            Push(&stack, expresion[i]); // Mete a la pila el caracter

        else if (expresion[i] == ')'){ // Si el caracter de la posición actual es = ')'
```

```
            while (!Empty(&stack) && Top(&stack) != '(') // Mientras la pila no este vacia y el tope no sea '('
                expresion[++k] = Pop(&stack); //En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)

            if (!Empty(&stack) && Top(&stack) != '(') //Si la pila no este vacia y el tope no sea '(' (despues de el ciclo)
                return -1; //El programa termina de manera incorrecta

            else // En caso de que no se cumpla
                Pop(&stack); // Saca el caracter de la pila
        }
        else {
            while (!Empty(&stack) &&                        //Mientras la pila no sea vacia
                Prec(expresion[i]) <= Prec(Top(&stack)))    // y la precedencia de el operador actual sea menor a la precedencia del tope
                expresion[++k] = Pop(&stack);    // En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)
            Push(&stack, expresion[i]);          // Metemos el caracter a la pila
        }
    }

    while (!Empty(&stack))      //Mientras la pila no sea vacia
        expresion[++k] = Pop(&stack ); //En el pre incremento (++k) se guarda el ultimo caracter de la pila (tope)

    expresion[++k] = '\0';      //Guarda el final la cadena utilizando el caracter nulo '\'
    printf( "\nEl resultado de la expresion es: \n %s", expresion);

    return 0;
}
```

Validar paréntesis

```
void cargarFormula(char *formula);


/*
    Cargar Formula (cargarFormula): recibe <- char *(formula);
    cargarFormula(formula)
    Efecto: Función para digitar la expresion y guardarla en una cadena "formula"
*/

void cargarFormula(char *formula)
{
    printf("Ingrese la formula:");
    fflush(stdin); // Limpia la entrada de el bufer
    gets(formula); // Guarda la cadena introducida en "formula"
    return;
}

int VerificarParentesis (pila *s, elemento *e){
    int f;

    for( f=0 ;f < strlen(e) ;f++ ){
        if (e[f] == '(' || e[f] == '[' || e[f] == '{' )
        {Push (s, e[f]);}

        else{
        if (e[f] == ')' ){
        if (Pop(s) != '(')
        return 0;
                }

        else if (e[f] == ']'){
        if (Pop(s) != '[')
        return 0;
        }

        else if (e[f] == '}'){
        if (Pop(s) != '{')
        return 0;
                }
            }
                    }

        if (Empty(s))
            return 1;
        else
            return 0;
    }
```

Bibliografía

"APLICACIONES DE LAS PILAS". prezi.com. https://prezi.com/rizwicqls7so/aplicaciones-de-las-pilas/#:~:text=Las%20pilas%20se%20utilizan%20en,la%20organización%20de%20la%20memoria. (accedido el 4 de diciembre de 2022).

"PILAS Y COLAS - yormiscpv". Google Sites: Sign-in. https://sites.google.com/site/yormiscpv/lenguaje-de-programacion-i/contenido/pilas-y-colas (accedido el 4 de diciembre de 2022).

"OPERACIONES CON PILAS Y APLICACIONES CON PILAS". Metodos de Programacion 1 UTEC. http://metodos1utec.blogspot.com/2012/10/operaciones-con-pilas-y-aplicaciones.html (accedido el 4 de diciembre de 2022).

"Pila (Estructura de datos) - EcuRed". EcuRed. https://www.ecured.cu/Pila_(Estructura_de_datos) (accedido el 4 de diciembre de 2022)

R. Henry. "¿Qué es una estructura de datos en programación? | Henry". Henry. https://blog.soyhenry.com/que-es-una-estructura-de-datos-en-programacion/#:~:text=En%20el%20ámbito%20de%20la,correcta%20para%20un%20determinado%20problema (accedido el 1 de diciembre de 2022).

D. Rodriguez. "Expresiones Infijas, posfijas y prefijas". prezi.com. https://prezi.com/5jq2pvrjfbng/expresiones-infijas-posfijas-y-prefijas/ (accedido el 1 de diciembre de 2022).