



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ALGORITMOS Y ESTRUCTURA DE DATOS

GRUPO: 2CV11

ALUMNO:

JONATHAN SAID GÓMEZ MARBÁN

PRÁCTICA 3A:

“PILA, OPERACIÓN DE INFijo A POSTFIJO”

Introducción

En esta práctica se busca pasar una operación infija a postfija, para ello debemos de tener presente que a notación de infijo es la notación común de fórmulas aritméticas y lógicas, en la cual se escriben los operadores entre los operandos en que están actuando, es decir, el orden es primer operando, operador, segundo operando, mientras que la notación postfija el orden es primer operando, segundo operando, operador.

Para resolver esta problemática de convertir una operación de infija a postfija se manejará el tema de pila o stack, la cual es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (último en entrar, primero en salir) que permite almacenar y recuperar datos.

Los conocimientos que debemos de tener presentes para poder trabajar con pilas son los apuntadores y estructuras (más en particular declaración de tipos definidos por el usuario), ya que se estarán utilizando durante el desarrollo del código, por lo que recordaremos como se usan. Podemos declarar un alias para un tipo de dato registro con la palabra reservada `typedef`, la cual proporciona un mecanismo para la creación de alias para tipos de datos primitivos (`int`, `char`, `float`, etc.). Por ejemplo:

```
typedef struct {  
    char c;  
    int i;} Ejemplo;
```

Después de haber hecho una estructura como la que tenemos a lado, podremos crear variables de tipo Ejemplo. Tal y como se muestra en la siguiente imagen. [Ejemplo a\[10\];](#)

Para poder trabajar con los valores de cada campo de la variable “`a`” de tipo Ejemplo, nos apoyaremos del el operador de dirección (`&`) y el operador de indirección (`*`), el primero de ellos regresa la dirección de una variable, mientras que el otro, toma la dirección de una variable y regresa el dato que contiene esa dirección. Para tener acceso a miembros de estructuras utilizamos el operador punto. El operador punto se utiliza colocando el nombre de la variable de tipo estructura seguido de un punto y seguido del nombre del miembro de la estructura. En caso de que la variable sea un apuntador entonces se tienen dos posibles sintaxis, la primera es `(*variable).campo` ó bien una mas común `variable->campo`.

Las funciones que definen o que se implementan cuando se va a ocupar una pila son dos operaciones básicas: apilar (`push`), que coloca un objeto en la pila, y su operación inversa, retirar (o desapilar, `pop`), que retira el último elemento apilado. Pero, en el programa se tienen mas funciones que complemetan y hacen mas eficiente el uso de las pilas como `initialize` (inicializa la pila), `isEmpty` (permite saber si la pila tiene elementos o no), `top` (que me da el valor del tope de la pila), `size` (que me da el tamaño de la pila).

Problemas relacionados al tema a los cuales te enfrentaste al programar la práctica.

En un principio fue difícil la implementación de la función push, ya que con la estructura planteada en clases y al momento de analizar los diagramas vistos en la misma, me confundía y hacía que el tope de la pila siempre apuntara a nulo, que en un principio nunca me di cuenta de este error y cuando empecé a programar, mi algoritmo estaba bien, pero lo que fallaba era la función push, porque hacía que el tope tuviera el valor de nulo y el programa no hacía nada.

También tuve problemas en un principio con la inicialización de algunas variables, ya que era algún campo de alguna estructura y además esa variable era apuntador, entonces tuve que investigar y encontré lo que puse en la introducción sobre el operador de dirección e indirección y eso me ayudó en toda el resto de practica a no tener mas esos errores.

Al momento de poner las condiciones para saber si a la pila había que hacerle un push o un pop, tenía que resolver el problema de la precedencia, que en un principio no tenía una idea clara, pero viendo el video de la clase y escuchando la explicación se me ocurrió y no tuve necesidad de buscar en internet, además de que creo es un código muy simple, pero resuelve la problemática.

En general, no tuve grandes problemas al resolver la práctica, solo algunos de sintaxis y lógica, pero que logré resolver en buen tiempo.

Documentación (capturas de pantalla).

```
1 //Funciones pila//
2 #include <stdio.h>
3 #include <string.h> //strlen
4 #include <stdlib.h> //malloc
```

Inclusión de archivos cabecera. Imagen 1.

Incluimos los archivos cabeceras que utilizaremos como lo son string.h y stdlib.h, además de la biblioteca estándar de entrada y salida, en los comentarios del programa se logra observar que funciones se ocupan de dichos archivos.

```
6 //Estructura de un nodo
7 typedef struct nodo{
8     char valor;
9     struct nodo *sig;
10 }Nodo;
```

Estructura de Nodo. Imagen 2.

Como se logra ver, solo tiene dos campos, uno de ellos es el valor de tipo char y el característico apuntador nodo, llamado sig, que nos permitirá tener enlazado a otro elemento.

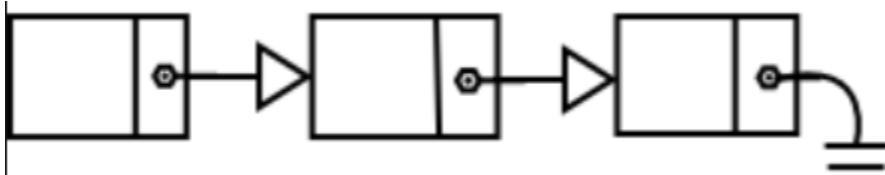


Diagrama de nodos. Imagen 3.

Ahora veremos la implementación de la pila, con las funciones ya mencionadas en la introducción, cada una de ellas ya está descrita en los comentarios del programa, que aparecerán en las capturas presentadas en este reporte.

```
18 //Inicializa una pila con tamaño 0 y el apuntador tope a nulo
19 void initialize (Pila *p){
20     p->size=0;
21     p->tope=NULL;
22 }
```

Función initialize (inicializar). Imagen 4.

```
24 //Si la pila está vacía regresa el valor 1 sino regresa un 0
25 int isEmpty(Pila *p){
26     int vacio=0;
27     if (p->tope==NULL)
28     {
29         vacio=1;
30     }
31     return vacio;
32 }
```

Función isEmpty. Imagen 5.

```
34 //Agrega un elemento a la pila y lo coloca en el tope de la misma
35 void push (Pila *p, Nodo e){
36     Nodo *pushing = (Nodo *) malloc(sizeof(Nodo));
37     pushing->sig = p->tope;
38     pushing->valor = e.valor;
39     p->tope=pushing;
40     p->size++;
41 }
```

Función push (apilar). Imagen 6.

Como vemos en esta función, creamos un apuntador a nodo y luego le reservamos un espacio de memoria, hacemos que apunte al nodo que se encuentre en el tope de la pila (la

que recibe la función), después modificamos su valor al valor que tenga el nodo que reciba la función y luego el tope de la pila será ese nodo que se ha creado y por último aumentamos el tamaño de la pila en 1

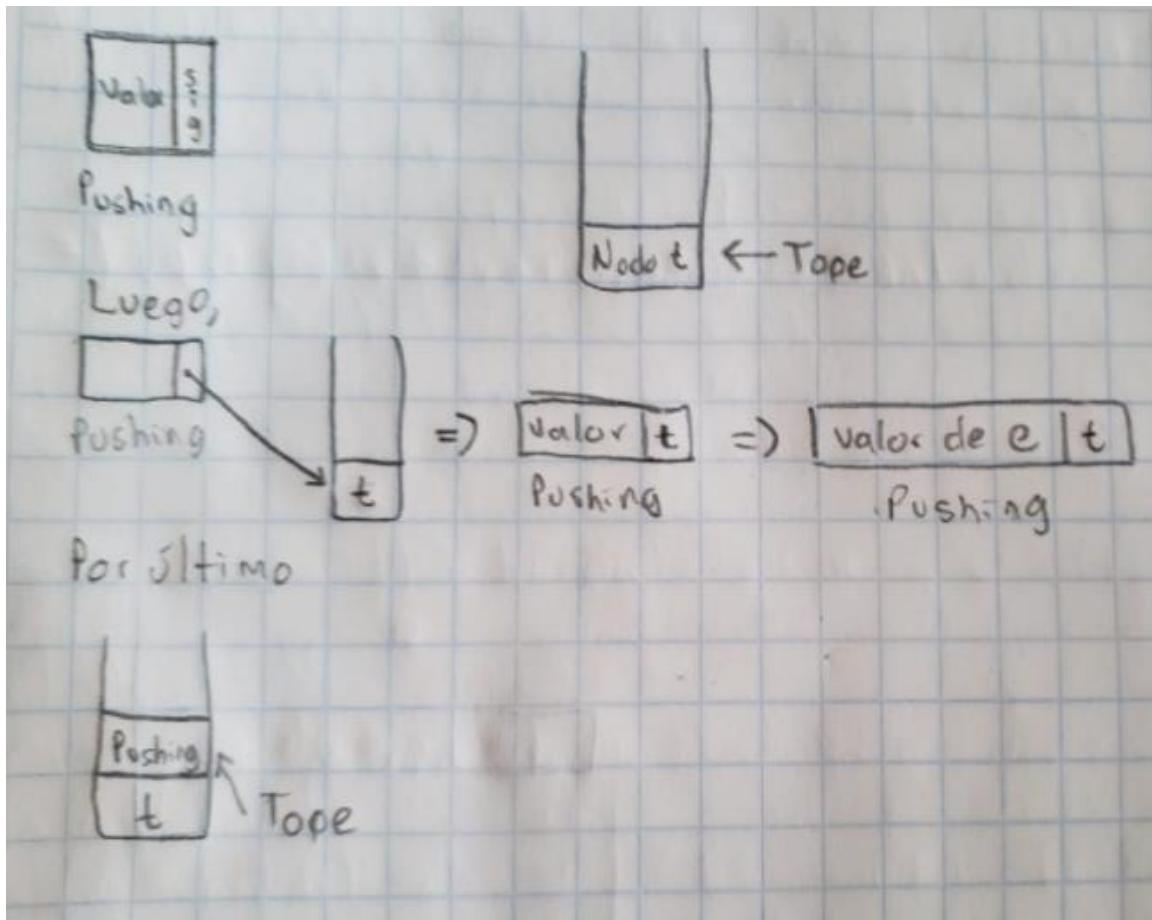


Diagrama de la función push. Imagen 7.

```

43 //Elimina el elemento que se encuentre en el tope
44 Nodo* pop (Pila *p){
45     if(isEmpty(p)==1){
46         return NULL;
47     }
48     Nodo *anterior_tope=p->tope;
49     p->tope=anterior_tope->sig;
50     anterior_tope->sig=NULL;
51     p->size--;
52     return anterior_tope;
53 }
```

Función pop. Imagen 8.

Como se ve, primero **se verifica si esta vacía la pila**, si esta vacía entonces regresa NULL, sino entonces se crea un nodo llamado anterior tope que se inicializa con los valores que tenga el nodo que se encuentre en el tope de la pila, luego, tomamos el valor del apuntador sig de "anterior tope" y lo ponemos como el nuevo tope de la pila y después al apuntador sig de ese nodo hacemos que apunte a nulo y hacemos que el tamaño de la pila decrezca en una unidad.

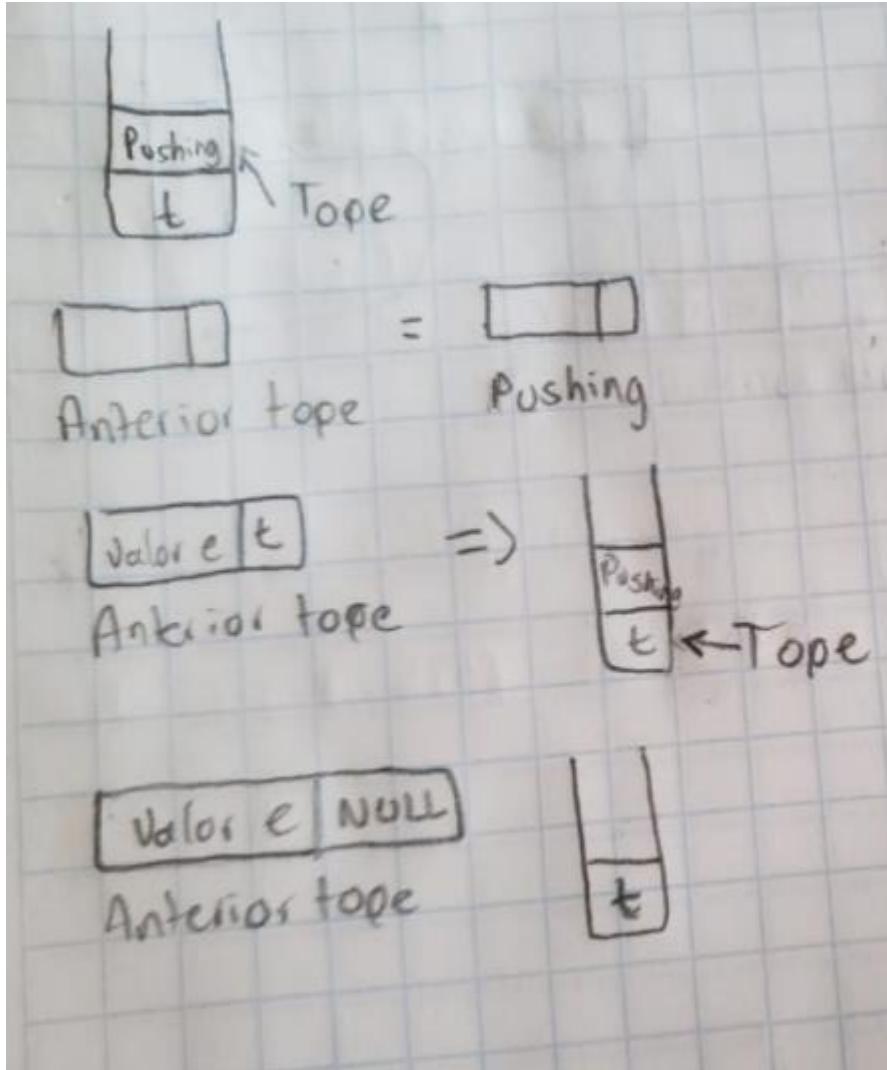


Diagrama de la función pop. Imagen 9.

```

55 //Indica el tope de la pila
56 Nodo* top(Pila *p){
57     return p->tope;
58 }
59
60 //Indica el tamaño de la pila
61 int size(Pila *p){
62     return p->size;
63 }

```

Funciones top y size. Imagen 10.

```

65 /*Recibe un carácter y me indica si es un operador o no, en caso de que lo sea regresa 0 sino entonces regresa 1*/
66 int esOperador(char operador){
67     int valor = 1;
68     if(operador == '+' || operador == '-' || operador == '*' || operador == '/' || operador == '^'){
69         valor = 0;
70     }
71     return valor;
72 }

```

Función para saber si es un operador. Imagen 11.

Esta función nos determina si lo que se recibe es un operador que se tenga que agregar a la pila, sabemos cuáles son dichos operadores, por lo que, simplemente se hacen comparaciones.

```

65 /*Analiza un carácter y dependiendo de este regresa un número, siendo el
66 simbolo de potencia el que tiene valor más alto, luego division y multiplicacion }
67 y por ultimo, resta y suma*/
68 int obtenerPrecedencia(char operador){
69     if (operador == '^'){
70         return 3;
71     }else if (operador == '/' || operador == '*'){
72         return 2;
73     }else{
74         return 1;
75     }
76 }

```

Función de precedencia. Imagen 12.

Esta función nos ayuda a saber como jerarquizar las operaciones, simplemente nos regresa un número más grande entre mayor precedencia tenga el operador, esto nos ayudará posteriormente al analizar los casos que podemos tener.

```
87 //Crea un nodo, modifica su valor y luego lo agrega a la pila
88 ~ void agregarValorNuevo(Pila* p, char operador){
89     Nodo e;
90     e.valor = operador;
91     push(p,e);
92 }
```

Función para reducir código (agrega nodos a la pila). Imagen 13.

Esta función, por simple que parezca (por cómo se describe en los comentarios del programa), es de mucha utilidad debido a que aparece en repetidas ocasiones dentro del código, lo que ayuda a lo que se tiene en tres líneas de código solo tenerlo en una sola.

```
94 /*Se extraen los valores del nodo del tope, se elimina el nodo y agrega el valor del
95  nodo eliminado a la cadena que se recibe como parametro en la posicion indicada*/
96 ~ void llevaPostfija (Pila *p, char* operador, int indice){
97     Nodo *e= pop(p);
98     operador[indice]=e->valor;
99     free(e);
100 }
```

Función para reducir código 2 (Borra y agrega a postfija). Imagen 14.

Esta función al igual que la anterior, permite reducir código, ya que aparece un par de veces dentro del main, aunque lo que realiza es sencillo (crea un apuntador a nodo donde se guarda el tope de la pila, luego el nodo del tope se elimina, se agrega el valor del nodo guardado a una cadena que en este caso será postfija y libera la memoria del nodo).

```

102 int main (){
103     char operacion[255];
104     char postfija[255];
105     Pila pila;
106     int contador=0;
107     initialize(&pila);
108     printf("Introduce la operacion: \n");
109     fgets(operacion, 256, stdin);
110
111     for (int i = 0; i < strlen(operacion)-1; i++){
112
113         //Aquí sabemos si es un operador o no
114         if(esOperador(operacion[i]) == 0){
115
116             //Si la pila esta vacía
117             if (isEmpty(&pila)){
118                 agregarValorNuevo(&pila, operacion[i]);
119
120             //Si el operador que se compara es igual al que se encuentra en la pila
121             }else if (operacion[i] == pila.tope->valor)
122             {
123                 postfija[contador]=operacion[i];
124                 contador++;
125
126                 //Si el operador que se quiere agregar es de mayor precedencia al que esta en la pila
127                 }else if (obtenerPrecedencia(operacion[i]) > obtenerPrecedencia(pila.tope->valor))
128                 {
129                     agregarValorNuevo(&pila, operacion[i]);
130
131                 //Si el operador que se quiere agregar es de menor o igual precedencia al que esta en la pila
132                 }else
133                 {
134                     while (isEmpty(&pila) == 0 && obtenerPrecedencia(operacion[i]) <= obtenerPrecedencia(pila.tope->valor))
135                     {
136                         llevaPostfija(&pila, postfija, contador);
137                         contador++;
138                     }
139
140                     agregarValorNuevo(&pila, operacion[i]);
141                 }
142
143             //Si no es un operador se agrega a la cadena postfijo directamente
144             }else
145             {
146                 postfija[contador]=operacion[i];
147                 contador++;
148             }
149         }
150     }
151
152     //Aqui vaciaremos lo que quede en la pila a postfija mientras no este vacía la pila
153     while (isEmpty(&pila) == 0)[
154         llevaPostfija(&pila, postfija, contador);
155         contador++;
156     ]
157
158     printf("%s \n",postfija);
159 }
```

Función main. Imagen 15.

Como se logra observar, en las primeras dos líneas de código declaramos dos arreglos de tipo carácter, el primero de ellos guardará la cadena que dé el usuario, mientras que la segunda será la cadena final que se imprimirá una vez finalizadas todas las condiciones.

Posteriormente creamos una pila y la inicializamos, creamos un contador que nos ayudará a manipular la posición de la cadena postfija y después preguntamos y recibimos la cadena con la que estará trabajando el programa.

Después con un ciclo for desde 0 hasta el tamaño de la cadena introducida, iremos iterando sobre la cadena de la operación dada por el usuario analizando sobre que caso puede caer. Cabe mencionar que el orden en el que están es de ayuda para no tener código de más o no tener que estar recorriendo más el código. Luego, podemos ver un if que me permite saber si es un operador o no, en cuestión de que el carácter que se está analizando no sea un operador se supondrá que es un operando y pasará directo a postfija, en la circunstancia contraria se tendrán distintos escenarios:

| Casos (orden en como aparecen en el programa) | Descripción del caso | Lo que realiza el programa |
|---|---|--|
| 1 | La pila este vacía | Se agrega a la pila sin preguntar nada |
| 2 | El valor del nodo tope de la pila sea igual al operador que se quiere agregar | El operador pasa directo a la cadena postfija |
| 3 | El operador que se quiere agregar tiene mayor precedencia que el que esta almacenado en la pila | Como esto si es permitido, entonces se apila el operador que se desea agregar |
| 4 | El operador que se desea agregar es de menor precedencia que el que se encuentra en el tope de la pila. | Lo primero que se realiza es hacer un pop a la pila, luego el valor que tenga el nodo al que se le acaba de hacer pop se va hacia la cadena postfija (esto se realiza las veces que sea necesario) es por ello que el while se evalúan dos cosas la precedencia y si la pila está vacía, ya que si lo está, el programa queda ciclado, posteriormente se agrega el operador a la pila. |

Tabla que representa los 4 casos que se tiene al tener un operador.

Por último, una vez que terminamos de recorrer toda la cadena (el for ha terminado su ejecución), se procede a vaciar la pila, todos los elementos que contenga serán tomados y agregados a la cadena postfija (recordando que las pilas son LIFO, es decir que el último elemento en entrar será el primero que agreguemos a postfija y así sucesivamente).

Ahora se procede a mostrar el ejemplo puesto en clase y otros más:

```
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>gcc 3A.c  
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>a  
Introduce la operacion:  
4+5*8-9  
458*+9-  
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>
```

Prueba 1 en consola. Imagen 16.

Se puede observar que es correcta esta operación (para cualquier duda, dentro de las bibliografías, la numero 4 es una calculadora que pasa de infijo a postfijo).

```
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>gcc 3A.c  
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>a  
Introduce la operacion:  
5+8*7^2-1  
5872**+1-  
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>
```

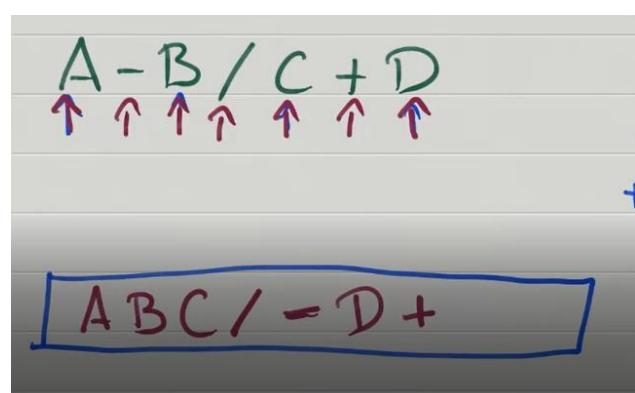
Prueba 2 en consola. Imagen 17.

Este fue uno de los ejemplos que me hizo darme cuenta de que el caso 4 tenía que tener un ciclo.

Finalmente haremos como prueba el ejemplo visto en clase:

```
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>a  
Introduce la operacion:  
A-B/C+D  
ABC/-D+  
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>
```

Prueba 3, ejemplo de la clase. Imagen 18.



Ejemplo que se vio en clase. Imagen 19.

Conclusiones

Con esta práctica se logró un mejor entendimiento y manejo de las listas LIFO o como simplemente se les puede llamar, pilas. A través de las pequeñas problemáticas que presenta pasar de una operación infija a postfija (que también es un conocimiento nuevo), pude poner a prueba las implementaciones que realicé y me di cuenta de que estaban mal hechas, pero investigando y haciendo diagramas pude encontrar la forma correcta de hacerlas. Ahora quedó completamente entendido como es que funciona una stack, sus funciones principales, el manejo de estas y un problema práctico en el que se pueda usar el tema. Entendí la importancia de los enlaces entre nodos y la lógica que conllevan. Además de que reforzé los conocimientos sobre los operadores de dirección e indirección.

Bibliografías

- [1] R. Cumplido (2015) "Pilas y Colas". Disponible en
<https://ccc.inaoep.mx/ingreso/programacion/corto2015/Curso-PROPE-PyED-5-Pilas-Colas.pdf>
- [2] (s.f.) "Apuntnadores" Disponible en
[http://www.utm.mx/~mgarcia/PE7\(Apuntadores\).pdf](http://www.utm.mx/~mgarcia/PE7(Apuntadores).pdf)
- [3] E. Quevedo. (2004, Mayo 16) "Tema 4.- Pilas y Colas" Disponible en
http://www.iuma.ulpgc.es/users/jmiranda/docencia/programacion/Tema4_ne.pdf
- [4] MathBlog (s.f.) "Infix / Postfix converter" Disponible en
<https://www.mathblog.dk/tools/infix-postfix-converter/>
- [5] (s.f.) "Principios de programación" Disponible en
<https://www.fing.edu.uy/tecnoinf/mvd/cursos/prinprog/material/teo/prinprog-teorico08.pdf>