INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ALGORITMOS Y ESTRUCTURA DE DATOS

GRUPO: 2CV11

ALUMNO:

JONATHAN SAID GÓMEZ MARBÁN

PRÁCTICA 3B:

“PILA, OPERACIÓN DE INFIJO A POSTFIJO (CON PARENTESIS)”

Introducción

Para esta práctica, se busca elaborar un programa similar al que se ha hecho en la práctica 3A, pasar de notación infija o notación postfija con la diferencia de poder utilizar paréntesis al escribir la operación infija. Esto cambia la lógica con la que es construida el programa, ya que, ahora, cada vez que tengamos un paréntesis de apertura, lo agregaremos a la pila sin siquiera ser necesario comparar la precedencia, cuando llegue un paréntesis de cierre se debe vaciar la pila hasta que se encuentre el '(' y, además, este último, no puede ser agregado a la cadena postfija. Recordemos que se tenían 4 casos a analizar cuando sabemos que se tiene un operador y que se tiene que hacer para agregar a la pila, en esta práctica se tiene 2 casos más, el cual es saber si el operador que se recibió es '(' o ')' y hacer las operaciones correspondientes.

Para la practica evidentemente haremos uso de apuntadores, estructuras y pilas, todo los conocimientos que se requieren saber para esta practica ya fueron dados en la introducción de la practica 3ª, así que solo daré un resumen.

Las pilas son conocidas como listas LIFO, esta es su característica principal, ya que, lo último que entra en la pila será lo primero en salir, para ello se ocupan funciones como push y por, que agregan o eliminan un elemento de la pila (nodo) respectivamente.

Recordemos también que las pilas son nodos enlazados uno con uno, el primero esta relacionado con el segundo, este a su vez con el tercero, mientras que el ultimo (el tope de la pila) estará apuntando a NULL. Respecto al tope de la pila es un elemento de este, que apunta al nodo mas reciente agregado.

También todo lo que ya hemos logrado hacer en a practica 3A para no partir de cero en esta. Se logro satisfactoriamente implementar todas las funciones propuestas en clase de la pila. Además, se crearon funciones que me permiten saber si recibo un operador o no, obtener precedencia del operador recibido, luego, también se creó la función que permite crear un nodo, modificar su valor y luego agregarlo a la pila en una sola función. Por último, se ha creado otra función la cual deja extraer el valor del nodo del tope, elimina dicho nodo y agrega el valor del nodo eliminado a la cadena que se recibe como parámetro en la posición indicada.

Para finalizar, sabemos que la implementación de la pila quedó sin modificación con respecto a la 3A por lo que creo que no será necesario mostrar las capturas de la implementación, ni explicarlas, ya que solo sería copiar y pegar y el objetivo es intentar explicar y hacer notar las diferencias de una practica con la otra.

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

En esta practica tuve un problema principal, el cual fue usar el while para repetir el pop hasta que pudiera agregar el elemento que quería agregar y respetando la jerarquía de las operaciones, fue difícil porque la consola no marcaba ningún error de sintaxis, por lo que mi error era lógico y pase bastante tiempo intentando encontrar la falla, la cual era que en la condición del ciclo (la precedencia), había veces que la pila se vaciaba y al momento de determinar la precedencia el programa se quedaba congelado y se salía, por lo que a este ciclo se le agrego la condición de que la pila no este vacía, conforme fui pensando esto, me di cuenta que también la práctica 3A requería esta modificación (ya que nunca pensé en esa situación), entonces son otra cosa que comparten estas 2 practicas sobre pilas.
Otro problema que tuve al hacer la practica fue modular el programa, ya que, en un principio trabajé la práctica 3A todo desde la main y como mucho de la practica 3B viene de la primera al tener el while y mas variables y casos a considerar fue un poco más difícil y tardado modular a comparación del anterior.
En general, esos fueron los dos problemas mas grandes que tuve, ya no fue tanto sobre pilas y su funcionamiento, porque al haber realizado y concluido la anterior, adquirí las bases y la lógica para manejar el tema.

Documentación (capturas de pantalla).

Como ya mencioné en la introducción de este trabajo, no quisiera copiar y pegar (incluso modificando algunas palabras o párrafos) de lo que ya se tiene y se sabe de la práctica 3A, por lo que solo haré un recuento de lo que se ocupo y es igual en ambas prácticas.

Las funciones de la implementación de la pila, como lo son initialize, isEmpty, push, pop, top y size, además de las estructuras de pila y nodo que también se ocuparon en la primera.

También las otras funciones que me sirvieron para trabajar en esta práctica, tales como esOperador (determinar si es un operador o no) en el cual se hacen ligeras modificaciones que se presentaran más adelante, obtenerPrecedencia (determinar una precedencia para cada operador) también con unas ligeras modificaciones, mientras que la función llevarPostfijo se mantiene intacta junto con agregarValorNuevo, recordemos que la primer de ellas ayuda a extraer los valores del nodo del tope, se elimina el nodo y agrega el valor del nodo eliminado a la cadena que se recibe como parámetro en la posición indicada, mientras que la segunda crea un nodo, modifica su valor y luego lo agrega a la pila.

```
65  int esOperador(char operador){
66      int valor = 1;
67          if(operador == '+' || operador == '-' || operador == '*' || operador == '/' || operador == '^' operador == '(' || operador == ')'){
68              valor = 0;
69          }
70      return valor;
71  }
```

Función esOperador (con modificaciones). Imagen 1}

Como vemos, solo aumentaron dos caracteres que tomaremos en cuenta para agregar a la pila, los cuales evidentemente son los paréntesis tanto de apertura, como de cierre.

```
/*Analiza un caracter y dependiendo de este regresa un numero, siendo el
simbolo de potencia el que tiene  valor mas alto, luego division y multiplicacion
depues, resta y suma y por ultimo, sino es nada de lo anterior regresa un 0*/
int obtenerPrecedencia(char operador){
    if (operador == '^'){
        return 3;
    }else if (operador == '/' || operador == '*'){
        return 2;
    }else if (operador == '+' || operador == '-'){
        return 1;
    }else{
        return 0;
    }

}
```

Función obtenerPrecedencia (con algunas adecuaciones). Imagen 2.

Como vemos, la modificación esta en que ahora los operadores + y – son los únicos indicados para recibir la precedencia 1, en caso de que no se cumpla ninguno de los primeros if's, estoy suponiendo que es un paréntesis (también se comprueba con la función "esOperador"), se le asigna un 0 para que al momento de tener que agregar cualquier otro operador arriba del paréntesis, se haga sin ningún problema.

Al igual que en la práctica 3A, tuvimos que crear una pila e inicializarla, un contador y dos arreglos de tipo carácter, uno que nos sirve para guardar lo que introduzca el usuario y la otra para ir agregando conforme corresponda (cadena postfija).

Como ya se explicó en la introducción, hay 2 casos más para analizar, los cuales son:

Casos agregados para problema con paréntesis	Descripción	Lo que se debe hacer para resolverlo
5	En caso de que haya un paréntesis de apertura	Este se debe agregar sin analizar que tenga la pila, sin importar su precedencia.
6	En caso de que haya un paréntesis de cierre	Se debe hacer pop e ir guardando los valores correspondientes en la cadena postfija hasta que se encuentre un paréntesis de apertura. En caso de que no se encuentre un paréntesis

		de apertura, entonces la pila se vaciara y al momento de hacer otro pop cuando la pila está vacía, entonces arrojará la validación del pop.

Tabla complementaria de casos posibles.

Aunque como casos se les haya puesto el 5 y 6, dentro del programa están acomodados de manera "estratégica" los casos, para así tener menos condiciones que recorrer o menos código que elaborar.

Primero tenemos si esta vacío, luego el de paréntesis de apertura, posteriormente el de cierre, prontamente si el operador que se quiere agregar es igual al que esta en el tope, después si tiene mayor precedencia el operador a agregar que el de la pila y por último (y como ya se indicó, cíclico) el caso en el que operador que se quiere agregar es menor al que esta en la pila.

Sino es un operador entonces pasa directamente a la cadena postfija. Una vez terminada de recorrer todo el arreglo donde esta almacenada introducida por el usuario, se procede a vaciar lo que queda en la pila, a partir de un ciclo, hasta que la pila quede vacía. En caso de que mientras se va haciendo el pop y guardando los valores en postfija se encuentre un paréntesis de apertura, se avisará que hay un error (el cual fue que hubo más paréntesis de apertura que de cierre). Esto lo sabemos porque como ya terminamos de recorrer la dada por el usuario, es imposible que haya un paréntesis de cierre, ya que esos ni siquiera se agregan a la pila. Y recordando también que no imprimirá la cadena si hay mas paréntesis de cierre que de apertura, por lo que ambos casos están cubiertos. Si todo se escribió de manera correcta, entonces el programa imprimirá la cadena en notación postfija y se dará por concluido el programa.

```c
int main (){
    char operacion[255];
    char postfija[255];
    Pila pila;
    int contador=0;
    initialize(&pila);
    printf("Introduce la operacion: \n");
    fgets(operacion, 256, stdin);

    for (int i = 0; i < strlen(operacion)-1; i++){
        //Verificamos si es un operador
        if(esOperador(operacion[i]) == 0){

            //Si la pila esta vacía, se agrega a la pila
            if (isEmpty(&pila)){
                agregarValorNuevo(&pila, operacion[i]);

            //Si el operador es un parentesis de apertura, se agrega a la pila
            }else if (operacion[i] == '('){
                    agregarValorNuevo(&pila, operacion[i]);

            //Si el operador es un parentesis de cierre, se hace pop hasta encontar parentesis de apertura
            }else if (operacion[i] == ')'){
                while (pila.tope->valor != '('){
                    llevaPostfija(&pila, postfija, contador);
                    contador++;
                }
                pop(&pila);

            //Si el operador de la cadena es igual al que esta en el tope de la pila, se agrega directo a la cadena postfija
            }else if (operacion[i] == pila.tope->valor){
                postfija[contador]=operacion[i];
                contador++;

            //Si el operador de la cadena es de mayor precedencia al que esta en la pila, se agrega a la pila
            }else if (obtenerPrecedencia(operacion[i])>obtenerPrecedencia(pila.tope->valor)){
                agregarValorNuevo(&pila, operacion[i]);

            //Si el operador de la cadena es de menor precedencia al que esta en la pila, se hace pop hasta que las condiciones s
            }else{
                while (isEmpty(&pila) == 0 && obtenerPrecedencia(operacion[i]) <= obtenerPrecedencia(pila.tope->valor)){
                    llevaPostfija(&pila, postfija, contador);
                    contador++;
                }
                agregarValorNuevo(&pila, operacion[i]);
            }

        //Si no es un operador, se pasa directo a postfija
        }else{
            postfija[contador]=operacion[i];
            contador++;
        }

    }
    /*Se vacía la pila (todos los elementos), se detecta si hay mas parentesis de apertura y se indica del fallo, en
    caso de que hubiera mas de cierre, entonces el programa no imprime nada debido a la validacion que se tiene en pop*/
    while(isEmpty(&pila) == 0){
        Nodo *de_la_pila=pop(&pila);
        if (de_la_pila->valor == '('){
            printf("Has puesto mas parentesis de apertura que de cierre");
            return 0;
        }else{
            postfija[contador]=de_la_pila->valor;
            contador++;
            free(de_la_pila);
        }
    }
    printf("%s \n",postfija);
}
```

Función main. Imagen 3.

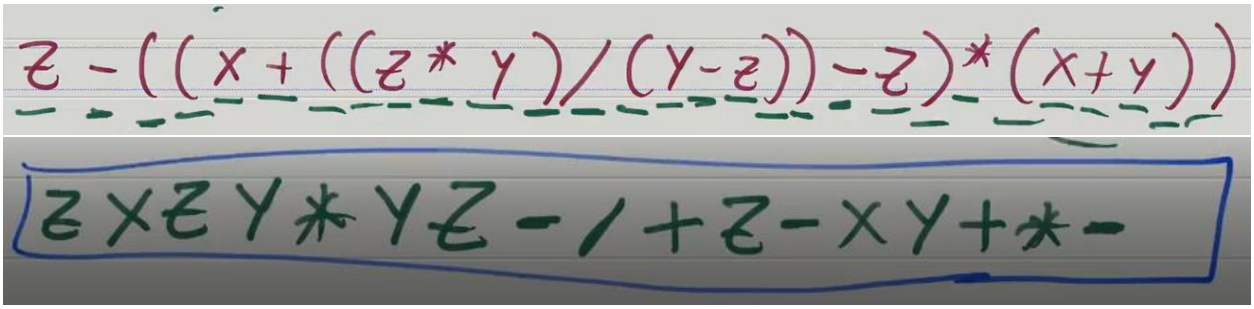
Ahora procederemos a presentar pruebas de que funciona el programa:

La primera que daremos es el ejemplo visto en clase:

```
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>gcc 3B.c

C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>a
Introduce la operacion:
z-((x+((z*y)/(y-z))-z)*(x+y))
zxzy*yz-/+z-xy+*-
```

Prueba 1, ejercicio visto en clase. Imagen 4.



De notación infija a postfija vista en clase. Imagen 5.

```
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>a
Introduce la operacion:
(3+5)*8
35+8*

C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>
```

Prueba 2. Imagen 6.

Como vemos es correcto, por último, haremos una con varios operadores y que la precedencia menor este entre paréntesis:

```
C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>a
Introduce la operacion:
9^2*10/(4+1)
92^10*41+/l

C:\Users\jona-\Documents\Jona\ESCOM\semestre 2\Estructura de datos\Pila>
```

Prueba 3. Imagen 7

Conclusiones

De esta práctica aprendí a tener más en cuenta las condiciones que se deben de tener para ejecutar algo, me costó mucho tiempo encontrar cual era el error, pero una vez que lo encontré se me hizo muy evidente, porque trataba de sacar precedencia a la basura que estuviera apuntando la pila cuando se quedaba vacía. Otra cosa importante que me llevo como aprendizaje de esta práctica es el uso de las pilas, con la primer practica sentí que había entendido de una buena manera, pero para esta segunda practica no me fue difícil manejar las pilas, entrar a los campos, modificar valores de los mismos, usar sin ninguna duda los operadores de dirección e indirección, sé que la practica trata de otro tema principal, pero sin duda pude aclarar dudas acerca de paso por valor y paso por referencia. Además, ya no se me hace tan difícil manejar este tipo de listas, que al principio encontré un poco difíciles de manipular.

Bibliografías

[1] M. León (2015) "pilas y colas y su implementación en object pascal". Disponible en http://www.informatica-juridica.com/wp-content/uploads/2019/02/Art%C3%ADculo-sobre-Pilas-y-Colas-converted.pdf

[2] (2009, feb. 12) "Manipulación de información. Pilas y colas" Disponible en https://www.youtube.com/watch?v=56jBvugTCn0

[3] G. Molto. (s.f.) "Tema 8- Implementación de Pila, Cola y Lista con Punto de Interés" Disponible en https://www.grycap.upv.es/gmolto/docs/eda/EDA_Tema_8_gmolto.pdf

[4] MathBlog (s.f.) "Infix / Postfix converter" Disponible en https://www.mathblog.dk/tools/infix-postfix-converter/