Programación de Operaciones Aritméticas con Polaca Inversa

Michael Vladimir Morales Molina, Diego Ignacio Paz Naula
Universidad de las Fuerzas Armadas ESPE
Sangolquí, Ecuador
mvmorales3@espe.edu.ec
dipaz@espe.edu.ec

Resumen— Este documento contiene el desarrollo de la codificación en c++ de la simulación de una calculadora para resolver operaciones algebraicas con las expresiones convertidas en "polaca inversa" para tener un mejor desarrollo y para respetar la jerarquía en los signos algebraicos.

Abstract--- This document contains the development of the coding in c ++ of the simulation of a calculator to solve algebraic operations with the expressions converted into "inverse polish" to have a better development and to respect the hierarchy in the algebraic signs.

Palabras Claves:

Polaca Inversa, Infija, Postfija, Prefija, Operaciones Aritméticas, Codificación, c++.

I. INTRODUCCION

En la antigüedad (1972) Heweltt Packard Company fue la primera empresa en elaborar una calculadora que permitía realizar operaciones aritméticas pero con su ingreso con una expresión en prefijo o en polaca inversa por lo cual se hacía posible respetar la jerarquía de los operadores en las expresiones, tomando este ejemplo se implementó un sistema informático que tiene como objetivo recibir una expresión aritmética introducida en forma infija (forma normal) y transformarla a polaca inversa (forma prefija, forma postfija) para con ello tener separado números de signos aritméticos para poder respetar la jerarquía de los mismos, para hacer esto posible se utilizan pilas o Stack en ingles que consiste en utilizar espacio de memoria según se vaya necesitando con lo que se debe ingresar una serie de datos pero estos se van ir ordenando uno tras otro por lo que se hace referencia a la normativa LIFO (Last Input First Output) o sus siglas en español (ultimo

en ingresar primero en salir) por lo que se debe tener una trasformación de expresiones, por ejemplo: «1+1» va a ser igual «1 1 +» y esta se encuentra en polaca inversa (forma postfija) y simultáneamente se va a transformar en «+ 1 1» que se encuentra de la misma manera en polaca inversa pero en su forma prefija.

II. Polaca Inversa

Para realizar la transformación de su forma infija a forma prefija y forma postfija se debe tener en cuenta que primero se debe procesar los signos que van a realizar las operaciones entre los datos para poder realizar esto se debe tener en cuenta que toda expresión matemática proviene de un árbol binario el cual es el que impone una estructura jerárquica en una colección de objetos.

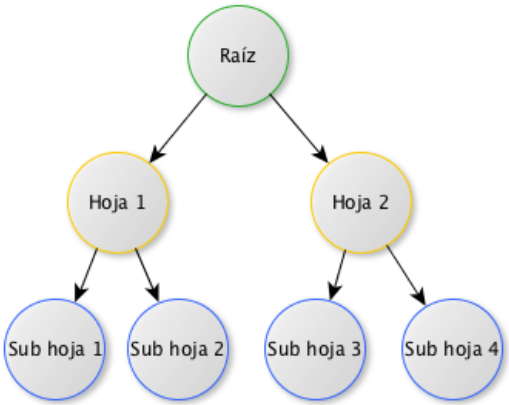

Fig. 1. Partes de un árbol binario

Para hacer la debida transformación a su forma prefija se debe seguir un algoritmo para recorrer el árbol el cual consiste en:

1. Procesar la raíz del árbol
2. Descender de nivel por su lado izquierdo y si se hallan raíces procesarlas
3. Descender y procesar las hojas
4. En el nivel más inferior regresar a la raíz y procesar por la parte derecha de la raíz
5. Regresar a la raíz inicial y proceder a hacer el mismo proceso por la parte derecha.

Ejemplo:

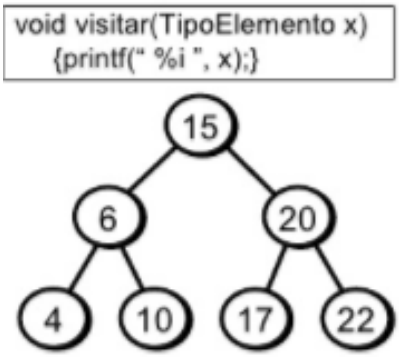

Fig. 2. Árbol binario.

Recorrido PreOrden (RID)

```
void preOrden(ArbolBinario raíz)
{  if(raiz)
   {    visitar(raiz–>dato);
        preOrden(raiz–>izq);
        preOrden(raiz–>der);
   }
}
```

El recorrido en PreOrden del árbol es el siguiente:
15, 6, 4, 10, 20, 17, 22

Fig. 3. Código para procesar un árbol binario en prefijo

Para hacer la debida transformación a su forma postfija se debe seguir un algoritmo para recorrer el árbol el cual consiste en:

1. Procesar las hojas del nivel más inferior, descendiendo por la parte izquierda.
2. Una vez procesada regresar a la raíz y descender por la parte derecha y procesar la hoja
3. Regresar a la raíz y procesarla.

```
void visitar(TipoElemento x)
   {printf(" %i ", x);}
```

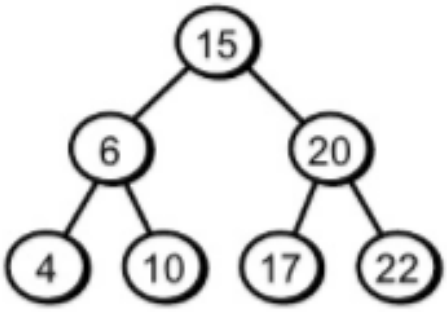

Fig. 4. Árbol binario.

Recorrido PostOrden (IDR)

```
void PostOrden(ArbolBinario raíz)
{  if(raiz)
   {    PostOrden(raiz->izq);
        PostOrden(raiz->der);
        visitar(raiz->dato);
   }
}
```

El recorrido en PostOrden del árbol es el siguiente:
4, 10, 6, 17, 22, 20, 15

Fig. 5. Código para procesar un árbol binario en prefijo

Para proceder a realizar la codificación en c++ se debe tomar en cuenta si se tiene o no paréntesis la expresión e ir almacenando tanto los paréntesis de la expresión como los signos de la misma en una pila, y en caso de ser datos (números) se ira poniendo en otra pila, si se encuentra en la expresión un signo y en la pila ya se encentra un signo ingresado, el signo de la pila se ingresa en la pila de los datos, así hasta terminar toda la expresión.

ENTRADA	PILA	SALIDA
5 ^ (6 * 8 / 23) – sin(45)		
^ (6 * 8 / 23) – sin(45)		5
(6 * 8 / 23) – sin(45)	^	5
6 * 8 / 23) – sin(45)	^ (	5
* 8 / 23) – sin(45)	^ (	5 6
8 / 23) – sin(45)	^ (*	5 6
/ 23) – sin(45)	^ (*	5 6 8
23) – sin(45)	^ (/	5 6 8 *
) – sin(45)	^ (/	5 6 8 * 23
– sin(45)	^	5 6 8 * 23 /
sin(45)	-	5 6 8 * 23 / ^
45	- sin	5 6 8 * 23 / ^
	- sin	5 6 8 * 23 / ^ 45
		5 6 8 * 23 / ^ 45 sin -

Tabla.1. Transformación de forma infija a postfija

III. Proceso y Operaciones

Como se explicó anteriormente, se procede a hacer la codificación del mismo en c++ para realizar la transformación de la forma infija a la forma postfija y a la forma prefija como se indica a continuación.

```
//-----Conversión de la
función a postfija
void conv_pos(char *ex,
char *epos, char **Simb,
char simb) {
int tope = -1;
int n = tama(ex);
char
*pila=(char*)malloc(1000*s
izeof(char));
while (*ex != '\0') {
simb = *ex;
rec_exp(ex);
n -= 1;
if (simb == '(') {
tope += 1;
*(pila+tope) = simb;
}
else {
if (simb == ')') {

while (*(pila+tope) !=
'('
{
int x = tama(epos);
*(epos+x) = *(pila+tope);
*(pila+tope) = '\0';
tope -= 1;
}
*(pila+tope) = '\0';
tope -= 1;
}
else {
if (sim(simb, Simb, epos)
== 0) {
int x = tama(epos);
*(epos+x) = simb;
}
else {
```

```
if (tama(pila)>0) {
while (prio(simb, *(pila+tope),
Simb) <= 0) {
int x = tama(epos);
*(epos+x) = *(pila+tope);
*(pila+tope) = '\0';
tope -= 1;
if (tope<0) {
break;
}
}
}
tope += 1;
*(pila+tope) = simb;
}
}
}
}
while (tope >= 0)
{
int x = tama(
epos);
*(epos+x) = *(pila+tope);
*(pila+tope) = '\0';
tope -= 1;
}
}
```

Fig. 6. Código para transformar de infijo a postfijo

```
//-----Conversión de la función a
prefija
void conv_pre(char *ex, char
*epre, char **Simb, char simb) {
      char *expre =
(char*)malloc(50 * sizeof(char));;
      limpiar(expre, 50);
      conv_pos(ex, expre, Simb,
simb);
      inver(epre, expre, simb);
}
```

Fig. 7. Código transformación de infijo a prefijo

```
//-----Verifica si es un operador
int sim(char s, char **Simb, char
*epos) {
int v = 0;
for (int i = 0; i<4; i++) {
for (int j = 0; j<2; j++) {
if (s == *(*(Simb+i)+j))
{
v = 1;
}
}
}
return(v);
}
```

Fig. 8. Código para verificar si es un signo matemático

Una vez que tenemos la expresión algebraica transformada en polaca inversa tanto en forma prefija y postfija se procede a realizar las operaciones aritméticas que se encuentren en la expresión, siendo así que respeta la jerarquía de operadores matemáticos, para ello se necesita extraer cada dato de la pila con su respectivo método getter con el cual se extraen los datos de forma LIFO de la pila y también se utiliza el método llamado pop, con el cual se libera la memoria del dato extraído, una vez extraídos los datos se inserta de nuevo el resultado de los datos operados con los siguientes códigos que se presenta a continuación en código c++.

```
float Pila::getnum(Pila *Nuevo)
{
    return Nuevo->num;
}
```

Fig. 9. Código getter para obtener el dato de la pila

```
Pila *Pila::pop(Pila *Nodo)
{
    if (Nodo != NULL)
    {
        Pila *temporal;
        temporal = Nodo;
        Nodo = Nodo->siguiente;
        temporal->dato = "";
        free(temporal);
    }
    else
    {
        printf("Lista Vacia");
        system("pause");
    }
    return Nodo;
}
```

Fig. 10. Código método POP libera espacio de la pila

```
Pila *Pila::push(Pila *Nodo, string dato)
{
    Pila *temporal = new Pila();
    temporal->dato = dato;
    temporal->siguiente = Nodo;
    Nodo = temporal;
    return Nodo;
}
```

Fig. 11. Código método PUSH inserta Datos a la pila

Para realizar las operaciones inscritas en la expresión se debe tener en cuenta que se tiene varios casos, de los diferentes signos matemáticos posibles para operarlos.

En el caso que se tenga un signo de sumatoria se procede a hacer lo siguiente.

```
case '+':
    val1 = numero->getnum(numero);
    numero = numero->pop(numero);
    val2 = numero->getnum(numero);
    numero = numero->pop(numero);
    numero = numero->push(numero, val1 + val2);
    break;
```

Fig. 12. Código "case" operador suma

Donde primero se extraen los valores de la pila y luego se procede a insertar el nuevo número calculado de la operación realizada en la pila con el método push.

En el caso que se tenga un signo de sustracción se procede a hacer lo siguiente.

```
case '-':
    val1 = numero->getnum(numero);
    numero = numero->pop(numero);
    val2 = numero->getnum(numero);
    numero = numero->pop(numero);
    numero = numero->push(numero, val2 - val1);
    break;
```

Fig. 13. Código "case" operador resta

Donde primero se extraen los valores de la pila y luego se procede a insertar el nuevo número calculado de la operación realizada en la pila con el método push.

En el caso que se tenga un signo de multiplicación se procede a hacer lo siguiente.

```
case '*':
    val1 = numero->getnum(numero);
    numero = numero->pop(numero);
    val2 = numero->getnum(numero);
    numero = numero->pop(numero);
    numero = numero->push(numero, val2 * val1);
    break;
```

Fig. 14. Código "case" operador multiplicación

Donde primero se extraen los valores de la pila y luego se procede a insertar el nuevo número calculado de la operación realizada en la pila con el método push.

En el caso que se tenga un signo de división hay que tener muy en cuenta que las divisiones para cero no existen por lo que se debe validar que el número dividendo sea diferente de cero por lo que se procede a hacer lo siguiente.

```
case '/':
    val1 = numero->getnum(numero);
    numero = numero->pop(numero);
    val2 = numero->getnum(numero);
    numero = numero->pop(numero);
    if (val1 == 0.0f) {
        printf("\nEXPRESION ALGEBRAICA MAL INGRESADA");
        system("pause");
        return 0;
    }
    numero = numero->push(numero, val2 / val1);
    break;
```

Fig. 14. Código "case" operador división

Donde primero se extraen los valores de la pila y luego se procede a insertar el nuevo número calculado de la operación realizada en la pila con el método push.

En el caso de que se tenga una función trigonométrica ya sea seno, coseno o tangente se procede a hacer lo anterior, con la diferencia que en este software se utilizaron series de Taylor para saber el valor de dichas funciones trigonométricas, como se muestra a continuación.

En el caso de ser seno.

```
case 's':
    cad = auxCad;
    strtok_s(cad, " ", &auxCad);
    trigo = seno(atof (cad), 100);
    numero = numero->push(numero, trigo);
    printf("\n%.2f", trigo);
    //cad = auxCad;
    //strtok_s(cad, " ", &auxCad);
    break;
```

Fig. 15. Código "case" función trigonométrica seno

En el caso de ser coseno.

```
case 'c':
    cad = auxCad;
    strtok_s(cad, " ", &auxCad);
    trigo = coseno(atof(cad),100);
    numero = numero->push(numero, trigo);
    //cad = auxCad;
    //strtok_s(cad, " ", &auxCad);
    break;
```

Fig. 16. Código "case" función trigonométrica coseno

En el caso de ser tangente se debe condicionar, debido a que la tangente es el seno sobre coseno, por lo que se debe contrarrestar que el coseno sea diferente de cero, por la explicación de que las divisiones para el número cero no existen.

```
case 't':
    cad = auxCad;
    strtok_s(cad, " ", &auxCad);
    trigo = coseno(atof(cad),100);
    if (trigo < 0.1)
    {
        printf("\nERROR SINTACTICO EN LA EXPRECION!!!");
        system("pause");
        system("cls");
        goto ingreso;
    }
    else
    {
        trigo1 = seno(atof(cad), 100);
        numero = numero->push(numero, trigo1 / trigo);
    }
```

Fig. 16. Código "case" función trigonométrica tangente

Como se indicó antes se usan Series de Taylor, por esa razón es que se utiliza funciones con el nombre de seno y coseno en los anteriores casos, a continuación el código en c++ de series de Taylor.

```
float seno(float x, int n)
{
    float seno = 0;
    if (n>0)
    {
        double signo = 1.0, p = Radianes(x), q = 1;

        for (int i = 0; i <= n; i++)
        {
            seno = seno + signo * (p / factorial(2 * i + 1));
            p = p * Radianes(x)*Radianes(x);
            signo = signo * (-1);
        }
    }
    return seno;
}
```

Fig. 17. Código Serie de Taylor para el seno

```
float coseno(float x, int n)
{
    float coseno = 0;
    if (n>0)
    {
        double signo = 1.0, p = Radianes(x), q = 1;

        for (int i = 0; i <= n; i++)
        {

            coseno = coseno + signo * (q / factorial(2 * i));
            q = q * Radianes(x)*Radianes(x);
            signo = signo * (-1);
        }
    }
    return coseno;
}
```

Fig. 18. Código Serie de Taylor para el coseno

Una vez realizado todas las operaciones extrayendo los números que están almacenados en la pila, se procede a mostrar por pantalla el resultado, de la siguiente forma.

```
Expresin ingresada: (((1+1)(2-2))+3(cos(45)))
La conversi¾n a Posfija es:  1 1 + 2 2 - * 3 cos 45 *
El resultado es: 2.12

Presione una tecla para continuar . . .
```

Fig. 19. Impresión salida por consola

IV. CONCLUCIONES

- Se ha concluido que para la transformación de formas, tanto a postfija como ha prefija se necesita de una pila para ir comparando los signos matemáticos (operadores), para así dar la prioridad a la jerarquía de los mismos.
- Se ha concluido que para poder operar en polaca inversa no es necesario la utilización de 2 pilas, pues en la transformación de formas ya se encarga de dar prioridad a la jerarquía de operadores.
- Podemos concluir que en este proyecto nos ayudo para la mejor comprensión de pilas y el manejo de las funciones de las mismas push y pop

V. REFERENCIAS

ElCapo2008. (24 de Enero de 2011). *SlideShare*. Obtenido de https://es.slideshare.net/elcapo2008/5-arboles-binarios

Sergio. (s.f.). *Arboles Binarios*. Obtenido de http://www6.uniovi.es/usr/cesar/Uned/EDA/Apuntes/TAD_apUM_04.pdf

Wikipedia. (17 de Enero de 2018). Obtenido de https://es.wikipedia.org/wiki/Calculadora