Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Campus Tecnológico Central de Cartago

[IC2001] Estructuras de Datos

Profesor: Ing. Víctor Garro Abarca

Manual de técnico

Calculadora

José Daniel Araya Ortega c.2022209303

Jocsan Adriel Pérez Coto c.2022437948

Entrega: 7/11/2022

II Semestre

Contenido

Estructuras de datos

Lista enlaza simple: Para contener de manera dinámica el número de posición de una pista del programa, se tiene que aplicar una lista enlazada simple, que nos permite agregar o eliminar dinámicamente los valores necesarios cuando el programa lo requiera, gracias a que trabaja con punteros y variables anónimas. Además de que esta se genera a base structs, lo que nos permite generar nuestros elementos con los parámetros que necesitemos, generando los nodos de la lista.

Un ejemplo de cómo se ve una lista enlazada es el siguiente:

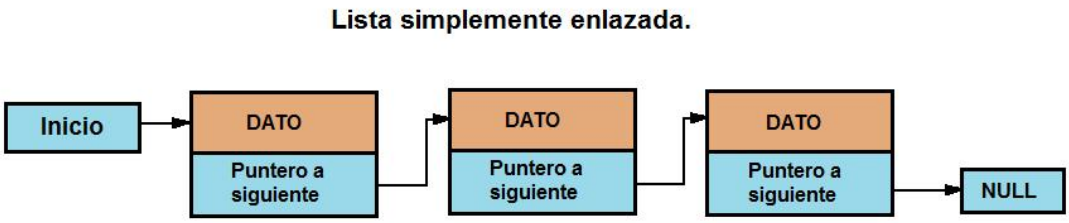


Fig 1. Lista enlazada simple. Tomado de: https://analisisyprogramacionoop.blogspot.com/2017/07/lista-simplemente-enlazada-C-sharp.html

Cola (FIFO): En este proyecto se implementará tanto una cola de elementos, donde se debe primero encolar, y luego desencolar de manera dinámica los elementos, como estructura que asemeja este comportamiento, pero no utilizan funciones de encolamiento y desenconamiento.

Para el caso de la cola completa esta estará vacía y con forme se le encolan elementos, se le irán también desencolando otros. Para comprender este concepto en la real tenemos:

Fig 2. Ejemplo real de FIFO. Tomado de:
https://www.freepik.es/fotos-vectores-gratis/fila-de-personas

También lo podemos ver como una lista enlazada:

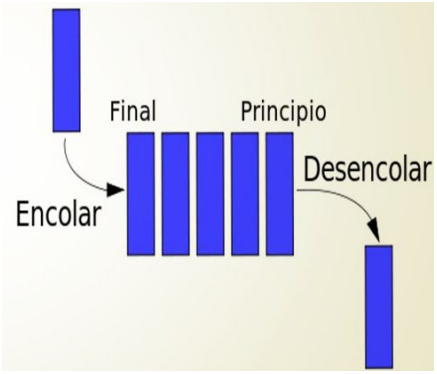


Fig 3. Ejemplo en LE de FIFO. Tomado de:
https://slideplayer.es/slide/16359303/

Por otro lado, este comportamiento se puede generar en listas estáticas donde se recorre la lista y en cada elemento se realiza alguna acción, donde al terminar de recorrer y volver a recorrer el primer en la lista será el primero en ser leído y el ultimo al final. Para utilizarlo y comprenderlo mejor hay que pensar en una fila circular, donde nadie sale y se recorre continuamente, como una lista enlazada circular. Recordando siempre que no es una lista enlazada de este tipo pero que se asemeja en comportamiento.

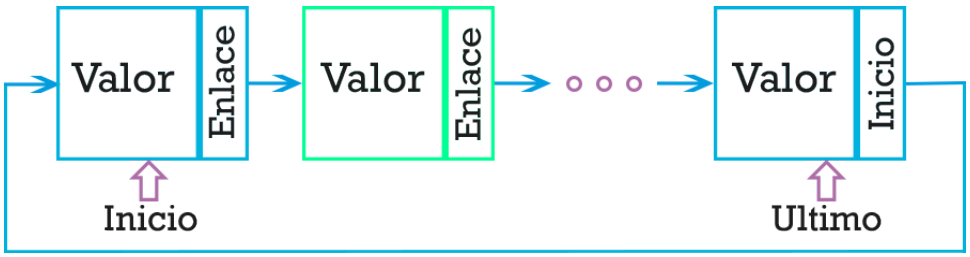
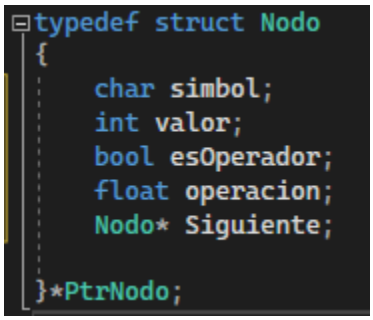


Fig 4. Ejemplo de estructura de LE circular. Tomado de:
http://codigolibre.weebly.com/blog/listas-circulares-simples-en-java

Nodos

Para el proyecto se utilizarán dos tipos nuevos de datos:

grupo: Nodo principal que se utilizara para todas las operaciones.



Los parámetros son:

- **char simbol:** Este es el char que en caso de ser un nodo operador se tomara en cuenta para el análisis.
- **int valor:** Este es el valor que se utiliza para cuando se trabaje con números, o se tenga que evaluar la prioridad en caso de ser un nodo operador.
- **int esOperados:** Este booleano lo utilizamos para saber si el nodo actúa como un operador o como un valor o símbolo deferente de estos.
- **bool operacion:** Debido a la presencia de divisiones se utilizará este parámetro en la evaluación de resultados.
- **Nodo* Siguiente:** Puntero del mismo tipo para la lista enlazada.

Relación entre estructuras

Nuestra lista enlazada simple se relacionará con los elementos de tipo PtrNodo, donde estos serán los nodos de esta. Esta lista en relación con el programa se utilizará como un registro dinámico de los valores ocupados en el programa.

Algoritmos/Lógica

Pilas que se utilizaran en los diversos algoritmos, siendo la de operadores, la de salida de expresiones y la que se utilizara para evaluar las instrucciones.

```
PtrNodo PilaOp;
PtrNodo PilaSalida;
PtrNodo PilaEval;
```

Manejo de listas enlazadas:

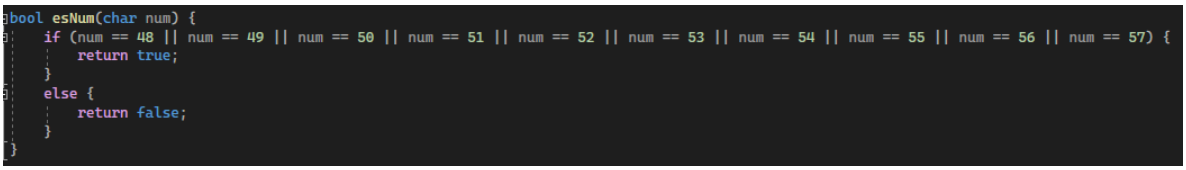
- Inicializa una lista enlazada, tomada de ejemplos antiguos.

```
void IniciarPila(PtrNodo& nodo)
{
    nodo = NULL;
}
```

- Destruye una lista enlazada, tomada de ejemplos antiguos.

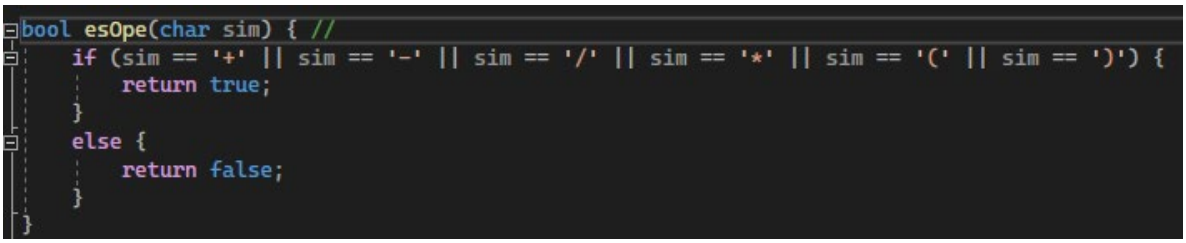
```
void DestruirPila(PtrNodo& nodo)
{
    PtrNodo Aux;
    Aux = nodo;
    while (Aux != NULL)
    {
        nodo = nodo->Siguiente;
        delete(Aux);
        Aux = nodo;
    }
}
```

esNum: Esta devuelve true si el char pasado corresponde a un número, según la tabla ASCII, false de lo contrario.



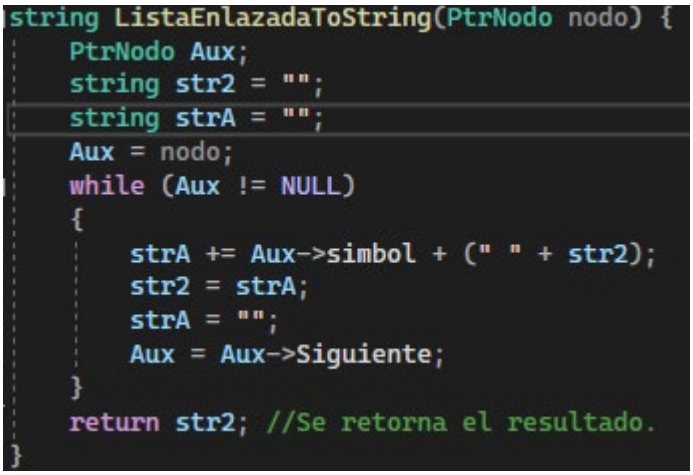
```
bool esNum(char num) {
    if (num == 48 || num == 49 || num == 50 || num == 51 || num == 52 || num == 53 || num == 54 || num == 55 || num == 56 || num == 57) {
        return true;
    }
    else {
        return false;
    }
}
```

esOpe: Esta devuelve true si el char pasado corresponde a un operador de la calculadora, según la tabla ASCII, false de lo contrario.



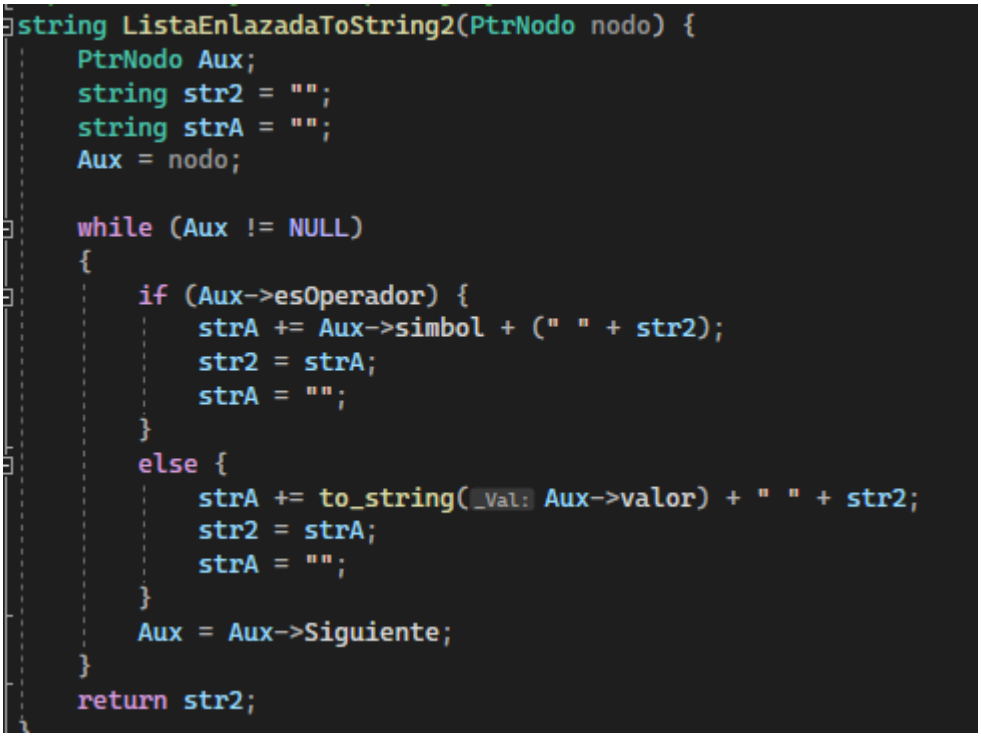
```
bool esOpe(char sim) { //
    if (sim == '+' || sim == '-' || sim == '/' || sim == '*' || sim == '(' || sim == ')') {
        return true;
    }
    else {
        return false;
    }
}
```

ListaEnlazadaToString: Toma una pila(lista enlazada), y devuelve como un string con espacios entre sus símbolos, esto se utilizará a la hora de convertir una expresión infija a una posfija. Esta se utiliza cuando la pila no tiene que tomar en cuenta valores numéricos.



```
string ListaEnlazadaToString(PtrNodo nodo) {
    PtrNodo Aux;
    string str2 = "";
    string strA = "";
    Aux = nodo;
    while (Aux != NULL)
    {
        strA += Aux->simbol + (" " + str2);
        str2 = strA;
        strA = "";
        Aux = Aux->Siguiente;
    }
    return str2; //Se retorna el resultado.
}
```

ListaEnlazadaToString2: Toma una pila(lista enlazada), y devuelve como un string con espacios entre sus componentes, esto se utilizará a la hora de convertir una expresión infija a una posfija y evaluar el valor de sus variables. Esta se utiliza tomando en cuenta que los nodos que no sean operadores se agregara su valor.



```
string ListaEnlazadaToString2(PtrNodo nodo) {
    PtrNodo Aux;
    string str2 = "";
    string strA = "";
    Aux = nodo;

    while (Aux != NULL)
    {
        if (Aux->esOperador) {
            strA += Aux->simbol + (" " + str2);
            str2 = strA;
            strA = "";
        }
        else {
            strA += to_string(_Val: Aux->valor) + " " + str2;
            str2 = strA;
            strA = "";
        }
        Aux = Aux->Siguiente;
    }
    return str2;
```

CrearNodo: Crea y retorna un nodo, donde si es operador o no se decidirá en base a char pasado. Si no es un operador se igualan los valores pasados y se declara esOperador como false.

```
PtrNodo CrearNodo(char sim, int val)
{
    PtrNodo nodo = new(Nodo);
    if (esOpe(sim)) {

        nodo->simbol = sim;
        nodo->esOperador = true;
        switch (sim) {
        case '+':
            nodo->valor = 1;
            break;
        case '-':
            nodo->valor = 1;
            break;
        case '/':
            nodo->valor = 2;
            break;
        case '*':
            nodo->valor = 2;
            break;
        case '(':
            nodo->valor = 0;
            break;
        }
    }
    else {

        nodo->simbol = sim;
        nodo->valor = val;
        nodo->esOperador = false;
    }

    nodo->Siguiente = NULL;
    nodo->operacion = 0;
    return nodo;
}
```

AgregarInicio: función que recibiendo dos referencias una de una lista y una de un nuevo objeto, agrega al inicio de la lista el objeto nuevo.

```
void AgregarInicio(PtrNodo& Lista, PtrNodo& Nuevo)
{
    Nuevo->Siguiente = Lista;
    Lista = Nuevo;
}
```

Push: Dada una pila y un objeto, hace un agregar al incio.

```
void Push(PtrNodo& nodo, PtrNodo Nuevo) {
    AgregarInicio(& Lista: nodo, &: Nuevo);
}
```

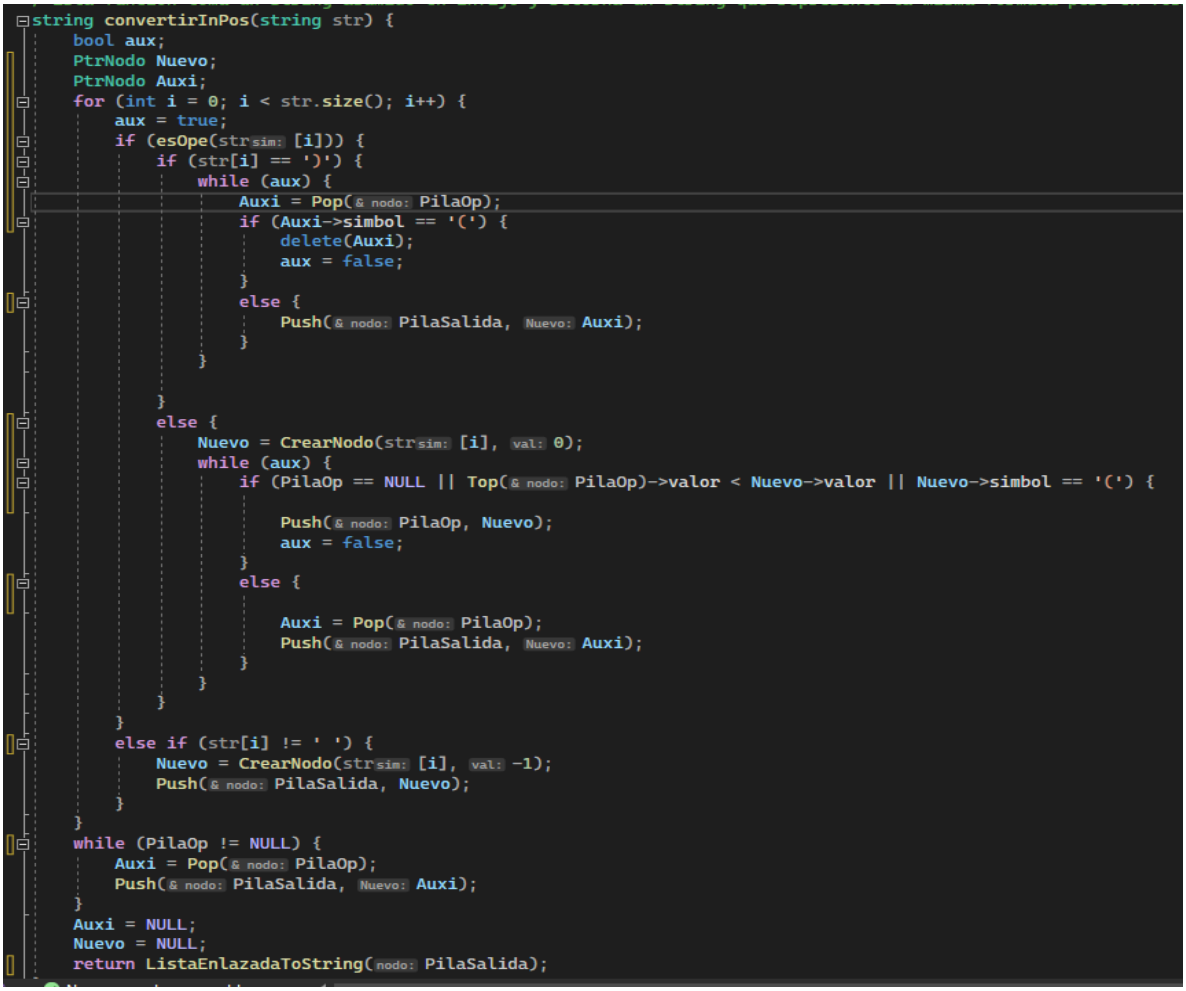
Top: Dada una pila, retorna, sin desligar, el primer nodo de la pila, el tope de esta.

```
PtrNodo Top(PtrNodo& nodo) {
    return nodo;
}
```

Pop: Dada una pila, retorna, desligando, el primer nodo de la pila.

```
PtrNodo Pop(PtrNodo& nodo) {
    PtrNodo Aux = nodo;
    nodo = nodo->Siguiente;
    Aux->Siguiente = NULL;
    return Aux;
}
```

covertirInPos: Esta función toma un string asumido en infijo y retorna un string que represente la misma fórmula, pero en formato posfijo. Se recorre el string paso por paso.

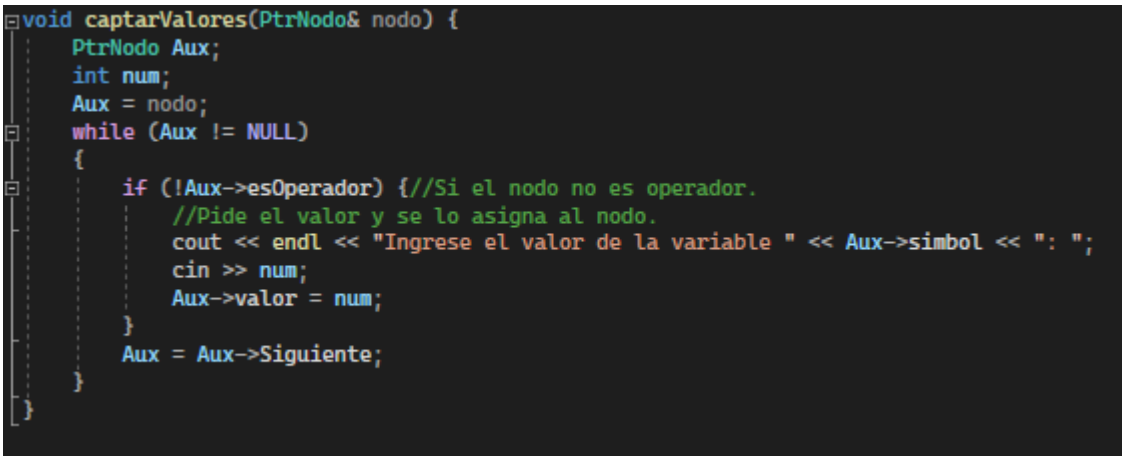


```
string convertirInPos(string str) {
    bool aux;
    PtrNodo Nuevo;
    PtrNodo Auxi;
    for (int i = 0; i < str.size(); i++) {
        aux = true;
        if (esOpe(str[i])) {
            if (str[i] == ')') {
                while (aux) {
                    Auxi = Pop(&PilaOp);
                    if (Auxi->simbol == '(') {
                        delete(Auxi);
                        aux = false;
                    }
                    else {
                        Push(&PilaSalida, Auxi);
                    }
                }

            }
            else {
                Nuevo = CrearNodo(str[i], 0);
                while (aux) {
                    if (PilaOp == NULL || Top(&PilaOp)->valor < Nuevo->valor || Nuevo->simbol == '(') {

                        Push(&PilaOp, Nuevo);
                        aux = false;
                    }
                    else {

                        Auxi = Pop(&PilaOp);
                        Push(&PilaSalida, Auxi);
                    }
                }
            }
        }
        else if (str[i] != ' ') {
            Nuevo = CrearNodo(str[i], -1);
            Push(&PilaSalida, Nuevo);
        }
    }
    while (PilaOp != NULL) {
        Auxi = Pop(&PilaOp);
        Push(&PilaSalida, Auxi);
    }
    Auxi = NULL;
    Nuevo = NULL;
    return ListaEnlazadaToString(PilaSalida);
```

EvaluacionPosFija: Esta función toma un string asumido en posfijo y evalúa sus valores, para devolver un valor de resultado

```
float EvaluacionPosfija(string str) {
    float res2 = 0;
    string str2 = "";

    PtrNodo Nuevo;
    PtrNodo Auxi1;
    PtrNodo Auxi2;
    for (int i = 0; i < str.size(); i++) {
        if (esOpe(str sim: [i])) {
            Auxi1 = Pop(& nodo: PilaEval);
            Auxi2 = Pop(& nodo: PilaEval);
            switch (str[i]) {
            case '+':
                res2 = (float)Auxi2->operacion + Auxi1->operacion;
                break;
            case '-':
                res2 = (float)Auxi2->operacion - Auxi1->operacion;
                break;
            case '/':
                if (Auxi1->operacion == 0) {
                    cout << "Error division por 0" << endl;
                    system( _Command: "Pause");
                    Push(& nodo: PilaEval, Nuevo: Auxi2);
                    Push(& nodo: PilaEval, Nuevo: Auxi1);
                    return NULL;
                }
                else {
                    res2 = (float)Auxi2->operacion / Auxi1->operacion;
                }
                break;
            case '*':
                res2 = (float)Auxi2->operacion * Auxi1->operacion;
                break;
            }

            Nuevo = CrearNodo(sim: '#', val: 0);
            Nuevo->operacion = res2;
            Push(& nodo: PilaEval, Nuevo);

            delete(Auxi2);
            delete(Auxi1);
            Auxi1 = NULL;
            Auxi2 = NULL;

        }
        else if (esNum(str num: [i])) {
            str2 += str[i];
            if ((str[i + 1] != NULL && !esNum(str num: [i + 1])) || str[i + 1] == NULL) {

                int number = stoi( _Str: str2);
                Nuevo = CrearNodo(sim: '#', val: number);
                Nuevo->operacion = number;
                Push(& nodo: PilaEval, Nuevo);
                str2 = "";
            }
        }
    }
    Nuevo = NULL;
    return Top(& nodo: PilaEval)->operacion;
}
```

captarValores: Esta función recibe una pila, y donde los nodos que no sean operadores pide por consola un valor para asignarles



```
void captarValores(PtrNodo& nodo) {
    PtrNodo Aux;
    int num;
    Aux = nodo;
    while (Aux != NULL)
    {
        if (!Aux->esOperador) {//Si el nodo no es operador.
            //Pide el valor y se lo asigna al nodo.
            cout << endl << "Ingrese el valor de la variable " << Aux->simbol << ": ";
            cin >> num;
            Aux->valor = num;
        }
        Aux = Aux->Siguiente;
    }
}
```

Main

Para esta parte solo se enfocará en las partes más importantes del main, como lo son el llamado a las funciones necesarias para la evaluación de la cifra indicada.

```
float resultado; //Auxiliar para resultado.
//Auxiliares para tratar con lso string con las expresiones
string str3;
string str;
string str2;
string str1;
```

```
//Se inicializan las pilas.
IniciarPila(& nodo: PilaOp);
IniciarPila(& nodo: PilaSalida);
IniciarPila(& nodo: PilaEval);
```

Se crea un ciclo de un menú con diferentes opciones

```
while (menu) {
    try {
        //Menu.
        system(_Command: "CLS");
        cout << " ********* Calculadora *********" << endl << endl;
        cout << "  1. Convertir de Infijo A Posfijo" << endl;
        cout << "  2. Evaluar una expresion Posfija" << endl;
        cout << "  3. Conversion y evaluacion de espresion." << endl;
        cout << "  4. Salir" << endl;
        cout << endl << "   Digite la opcion: ";
        cin >> opc;//Se toma la opcion digitada.
```

opc: va a cambiar según lo que se indique

Caso 1: luego de indicar el numero 1, se llama a la función de convertirInPos

```
case 1: {
    system( _Command: "CLS");
    cout << "---#Conversion de Infijo A Posfijo#---" << endl;
    cout << "Ingrese la expresion Infija: ";
    cin >> str;
    str3 = convertirInPos(str);
    cout << endl << "Expresion Posfija: " << str3 << endl;
    system( _Command: "Pause");
    DestruirPila( & nodo: PilaOp);
    DestruirPila( & nodo: PilaSalida);
    break;
```

Al final, se va a entrar un resultado según lo que indique el usuario.

Caso 2: luego de indicar el número 2, se llama a la función de EvaluacionPosFija

```
case 2: {
    system( _Command: "CLS");
    cout << "---#Evaluacion de Posfijo#---" << endl;
    cout << "Ingrese la expresion Posfija: ";
    cin >> str2;
    resultado = EvaluacionPosfija(str2);
    system( _Command: "Pause");
    cout << endl << "Resultado de la expresion: " << resultado << endl;
    system( _Command: "Pause");
    DestruirPila( & nodo: PilaEval);
    break;
```

Al final, se va a entrar un resultado según lo que indique el usuario.

Caso 3: luego de indicar el número 3, se llama a la función de convertirInPos

```
case 3: {

    system( _Command: "CLS");
    cout << "---#Conversion y evaluacion#---" << endl;
    cout << "Ingrese la expresion Infija: ";
    cin >> str;
    str3 = convertirInPos(str);
    cout << endl << "Expresion resultante: " << str3 << endl;
    system( _Command: "Pause");
    //Despues se toma la pila de salida y se capturan los valores de las variables dentro de la misma.
    captarValores( & nodo: PilaSalida);
    //Ya con los valores, se convierte la pila de salida a un string.
    str1 = ListaEnlazadaToString2( nodo: PilaSalida);
    //Y se evalua para dar el resulado, imprimirlo y destruir las pilas usadas.
    resultado = EvaluacionPosfija(str1);
    cout << endl << "Resultado de la expresion: " << resultado << endl;
    system( _Command: "Pause");
    DestruirPila( & nodo: PilaOp);
    DestruirPila( & nodo: PilaSalida);
    DestruirPila( & nodo: PilaEval);
    break;
```

En el caso 3, luego se van a pedir los valores necesarios según lo que indique el usuario.

Al final de cada caso siempre se va a destruir la pila, ya que esta no se va a necesitar mas y se necesita que este limpia para otro caso

Caso 4: Funciona para salirse del programa.