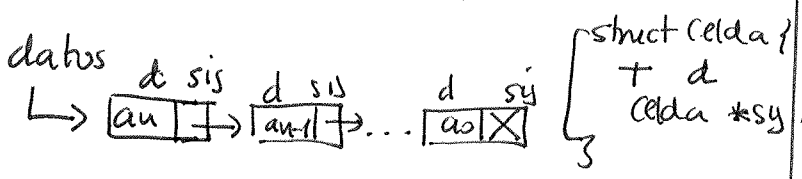
LECCION 8

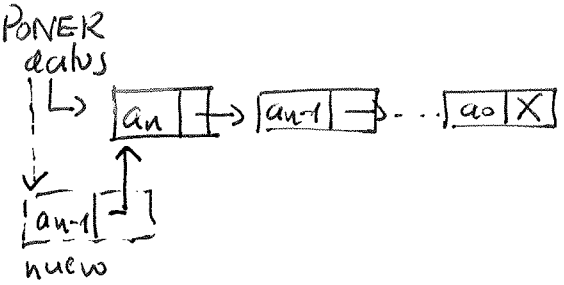
PILAS: IMPLEMENTACIÓN BASADA en CELDAS ENLAZADAS



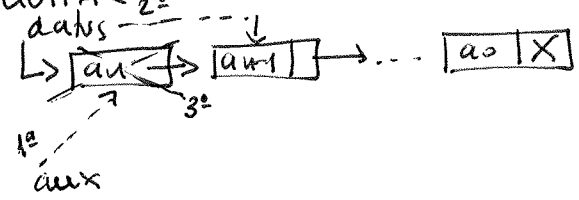
OPERACIONES

Tope = Consulta datos → d

PONER



QUITAR




```
#ifndef __PILA_H
#define __PILA_H
template <class T>
struct Celda{
   T d;
   Celda *sig;
};
template <class T>
class Pila {
  private:
    Celda<T> *primera;
    void Copiar(const Pila<T> &P);
    void Borra();
  public:
    Pila();
    Pila(const Pila<T> &P);
    Pila<T> operator=(const Pila<T> &P);
    ~Pila();
    T Tope() const;
    void Quitar();
    void Poner(const T &v);
    int size() const;
    bool Vacia() const;
};
#include "Pila.cpp"
#endif
```

```
// Pila.cpp
template <class T>
void Pila<T>::Copiar(const Pila<T> &P){
   if (P.primera==0) // es vacía
      primera=0
   }
   else{
      primera=new Celda<T>;
      primera->d=P.primera->d;
      Celda<T>* p=primera;
      Celda<T>* q=P.primera->sig;
      while (q!=0){
         p->sig=new Celda;
         p=p->sig;
         p->d=q->d;
         q=q->sig;
      }
      p->sig=0;
   }
}
template <class T>
void Pila<T>::Borra(){
   while (primera!=0){
      Celda<T>* aux=primera;
      primera=primera->sig;
      delete aux;
   }
}
```

LECCION 8. continuación

```
template <class T>
Pila<T>::Pila (){
    primera=0;
}

template <class T>
Pila<T>::Pila(const Pila<T> &P){
    copiar(P);
}

template <class T>
Pila<T>::~Pila(){
    Borrar();
}

template <class T>
T Pila<T>::Tope() const{
    assert(primera!=0);
    return primera->d;
}

template <class T>
void Pila<T>::Quitar(){
    assert(primera!=0);
    Celda *aux=primera;
    primera=primera->sig;
    delete aux;
}

template <class T>
void Pila<T>::Poner(const T &v){
    Celda* aux=new Celda;
    aux->d=v;
    if(primera==0){
        primera=aux;
        primera->sig=0;
    }else{aux->sig=primera;
        primera=aux;
    }
}
```

```
template <class T>
int Pila<T>::size() const{
    int cnt=0; Celda* p=primera;
    while(p!=0){
        cnt++;
        p=p->sig;
    }
}

template <class T>
bool Pila<T>::vacia() const{
    return primera==0;
}
```

EFICIENCIA

	Pila como Vector	Pila como celdas Enlazada
Poner	O(n) / Amortizado O(1)	O(1)
Quitar	O(n) ← Amortizado	O(1)
Vacia	O(1)	O(1)
Tope	O(1)	O(1)

Ejemplo de uso

```
#include "Pila.h"
int main(){
    Pila<int> P;
    int a; cin>>a;
    while(a>0){
        P.Poner(a);
    }
    while(!P.vacia()){
        cout<<P.Tope();
        P.Quitar();
    }
}
```

3.

LECCION 8

STACK (Pila en STL)

LIBRERIA ⟹ #include <stack>

CLASE ⟹ class stack (clase plantilla)

Operaciones

- emplace, empty, pop, push, size, swap, top

EJEMPLO

```
#include <stack>
using namespace std;
int main(){
    stack<string> ps1,ps2;
    ps1.emplace("Primero");
    ps2.emplace("Segundo");
    ps1.swap(ps2);
    ps2.emplace("Tercero");
    cout<< "Elementos en ps2 " <<ps2.size();
    while(!ps2.empty()){
            cout<<ps2.top();
            ps2.pop();
     }
}
```

EJERCICIOS.- Hacer una funcion que dada una sentencia que puede induir parentesis, corchetes y aritmética llaves analice si es correcta.

p.e [(3+5])+2 no es correcta.

```
#include <stack>
using namespace std;
int is_symbol( char c, char *especial, int n){
    for (int i=0; i<n; i++)
        if (c == especial[i])
            return i;
    return -1;
}
bool is_correct ( const string &exp ...){
    char open_esp[3] = {'(', '{', '['};
    char close_esp[3] = {')', '}', ']'};
    stack <char> mipila;
    for (int i=0; i<exp.size() ...; i++){
        if (is_symbol(exp[i], open_esp, 3) != -1)
            mipila.push(exp[i]);
        else
        int aux= is_symbol(exp[i], close_esp, 3);
            if (aux != -1){
                if (mipila.empty()) return false;
                char oo = mipila.top(); mipila.pop();
                int aux2 = is_symbol(oo, open_esp, 3);
                if (aux1 != aux2)
                    return false;
            }
    }
    if (mipila.empty() == false ) return false
    else return true;
}
```

LECCION 8

Notacion Polaca (N.P)

es una notacion algebraica donde primero aparece los operandos y luego el operador

N.P

$3+4 \Longrightarrow 3\ 4\ +$

VENTAJAS: No necesita parentesis para expresar la prioridad

$3+(4.7) \Longrightarrow 3\ 4\ 7\ .\ +$

$(3+4).7 \Longrightarrow 3\ 4\ +\ 7\ .$

¿Como evaluar la notacion polaca? ⇒ Usando una Pila

NP

NOTACION INFIJA

$3+((2+4)*5)-1 \Longrightarrow 3\ 2\ 4\ +\ 5\ *\ +\ 1\ -$

ENTRADA	OPERACION	PILA
3	Poner(3)	3
2	Poner(2)	3 2
4	Poner(4)	3 2 4
+	sumar	3 6
5	Poner(5)	3 6 5
*	Multiplicar	3 30
+	sumar	33
1	Poner(1)	33 1
-	resta	32

Suponer que queremos implementar un editor de texto muy básico. El modo de funcionamiento sería escribe el usuario una frase. Si le da a intro el programa saca todo lo escrito hasta el momento por el usuario y espera que le de otra frase para añadirla a las anteriores. Pero si el usuario pulsa la tecla ESC la ultima frase que se introdujo no se tiene en cuenta, se saca por pantalla lo introducido sin esta frase. Este proceso es como implementar la operación Undo. Para implementar este progama usar una Pila de forma que cada entrada conforma lo anterior mas la nueva frase introducida.

□

Ejemplo 4.2.3

En **notación Polaca o notación Postfijo** las operaciones aritméticas se describen de forma diferente a como las escribimos en notación infijo: (operador izquierda *operador* operando derecha). En la notación Polaca primero van los operandos y detrás los operadores. Por ejemplo si tenemos a+b en notación Polaca sería ab+. Otro ejemplo sería si tenemos en infijo 3 - 4 + 5 en notación Polaca sería 3 4 - 5 +. Usando una pila podemos evaluar una expresión en notación polaca, suponiendo que los operadores son +,-,/,* y los operandos son valores enteros.

```
1  #include "Pila.h"
2  #include <string>
3  #include <sstream>
4  #include <iostream>
5  using namespace std;
6  void QuitarBlancos(string &expresion){
7    while (expresion.size()>0 && expresion[0]==' ')
8         expresion= expresion.substr(1,str::npos);
9  }
10
11
12 bool Operador(string &expresion, char &operador){
13   QuitarBlancos(expresion);
14   if (expresion.size()>0){
15    if (expresion[0]=='+' || expresion[0]=='-' ||
16     expresion[0]=='*' || expresion[0]=='/')
17        operador = expresion[0];
18        expresion= expresion.substr(1,str::npos);
19        return true;
20    }
21   }
22   return false;
23 }
24
25 void GetOperando(string & expresion,int &operando){
26
27     QuitarBlancos(expresion);
```

```
28      if (expresion.size()>0){
29       stringstream ss;
30       string aux;
31
32       ss.str(expresion);
33       ss>>aux; //hasta el primer separador
34       //le quitamos a expresion lo leido en aux
35       expresion=expresion.substr(aux.size(),str::npos);
36       //convertimos de string a int
37       ss.str(aux);
38       ss>>operando;
39      }
40  }
41  int main(){
42  string expresion;
43  cout<<"Introduce una expresion (con separadores espacio en blanco):";
44  cin>>expresion;
45  Pila<int>mipila;
46  while (expresion.size()>0){
47     char operador;
48     if (Operador(expresion,operador)){ // si lo que hay es un operador
49         //sacamos de la pila los dos operandos
50         //y operamos y el resultado se pone en la pila
51         int op2=mipila.Tope();
52         mipila.Quitar();
53         int op1=mipila.Tope();
54         mipila.Quitar();
55         switch(operador){
56           case '+': int r=op1+op2;
57                     mipila.Poner(r);
58                     break;
59           case '-': int r=op1-op2;
60                     mipila.Poner(r);
61                     break;
62           case '*': int r=op1*op2;
63                     mipila.Poner(r);
64                     break;
65           case '/': int r=op1/op2;
66                     mipila.Poner(r);
67                     break;
68     }
69     else {//Operando
70       int operando;
```

```
71        GetOperando(expresion,operando);//obtiene de expresion el operando
72        mipila.Poner(operando);
73      }
74    }
75    cout<<"El resultado es: "<<mipila.Tope();
76  }
```

□

Ejercicio 4.2
Usando el T.D.A Pila escribir un programa que convierta un notación infija a notación postfija o Polaca.
□

4.3 Cola

1. **Especificación**: Son estructuras de datos lineales que contienen una secuencia de datos

   $$\{a_0, a_1, \cdots, a_n\}$$

   Y están especialmente diseñadas para hacer las inserciones por un extremo y los borrados y consultas por otro. El extremo en el que están el primer elemento $(a_0, a_1, \cdots)$ se llama *frente*, y es por el que se hacen las consultas y borrados. El extremo en el que están los últimos valores (a_n) se llama *última* y es por el que se realizan las inserciones. Las colas responden a la política FIFO (*First Input First Output*).
2. **Operaciones típicas**:
   - *Frente* $\longrightarrow$ consulta o accede al elemento en el frente
   - *Vacía* $\longrightarrow$ devuelve true si la cola está vacía
   - *Quitar* $\longrightarrow$ elimina el elemento que está en el frente
   - *Poner* $\longrightarrow$ añade un nuevo elemento por el final (la posición última).
3. **Implementación**:
   - Con celdas enlazadas y dos punteros (todas las operaciones nos cuestan $O(1)$)
   - Con vectores

4.3.1 Celdas enlazadas y dos punteros

```
1  //cola.h
2  #ifndef _COLA_H
3  #define _COLA_H
4
5  template <class T>
6  class Cola {
7  private:
8      //Fuera del entorno de la clase, no existe celda
9      //para que exista, lo definimos o bien fuera o en la parte publica
10     struct Celda {
```

EJERCICIO Pasar una expresión en notación infija a notación postfija

ENTRADA: Una cadena con la expresión infija
SALIDA: Una cadena con la expresión postfija - out
DATOS LOCALES: Pila que contiene operadores y paréntesis izquierdos

3 + ((4 * 7) - 2)

ENTRADA	OPERACION	OUT - SALIDA	PILA
3	Insertar al final de out	3	
+	Inserta + en la PILA		+
(	Inserta (Poner('(') en la PILA		+ (
(	Poner('(')		+ ((
4	Insertar al final de out	3 4	
*	Poner('*')		+ ((*
7		3 4 7	
)	Sacar de la Pila si es operador ponerlo en OUT	3 4 7 *	+ ((
	Volver a sacar ~~y si es (~~ hasta obtener (		+ (
)		3 4 7 * +	+ (
-	Poner('-')		+ (-
2		3 4 7 * + 2	
)		3 4 7 * + 2 - +	