Algoritmos
Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán
2023

PILA-STACK(2)

Notaciones

La forma tradicional de escribir expresiones aritméticas es la denominada notación ***infija***, en la cual el operador se escribe entre los operandos como en este ejemplo :

A + B

Existen otras formas de escribir expresiones aritméticas como la:

notación *prefija* o polaca :

+ A B

o la **notación *postfija* o polaca inversa (Reverse Polish Notation**, o **RPN):**

A B +

Notaciones

- La denominación polaca se debe al matemático de dicho origen de nombre **Jan Lukasiewicz**, quien en la década del 50 realizó importantes estudios sobre las propiedades de estas notaciones.

Las notaciones pre y postfija tienen como **ventaja** sobre la notación infija tradicional que:

- al evaluar una expresión escrita en pre o postfija, no es necesario especificar *precedencia de operadores*,
- cualquier expresión en pre o postfija puede ser escrita sin la necesidad de usar *paréntesis*.

Ejemplos: Notaciones

Expresión aritmética en:

Notación infija	Notación prefija	Notación postfija
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

PILA
Tratamientos de expresiones aritméticas

1) Algoritmo CONVERTIR (usa pila de operadores)

- Entrada: Expresión en notación infija
- Salida: Expresión en notación postfija

2) Algoritmo EVALUAR (usa pila de operandos)

- Entrada: Expresión en notación postfija instanciada
- Salida: Valor de la expresión

PILA
Aplicación: Convertir infija a postfija

- Muchos compiladores transforman primero la expresión infija en una postfija en la cual el operador sigue al operando, y entonces generan instrucciones de maquina para evaluar esta expresión postfija.

- Este proceso se usa porque la transformación de infija a postfija es directa, y una expresión postfija es más fácil de evaluar mecánicamente que una expresión infija.

- Ejemplo:

  Infija: 2 + 3 * 5 postfija: 2 3 5 * +

PILA
Aplicación: Convertir infija a postfija

Escribir un **algoritmo** en términos del **ADT PILA** para **CONVERTIR** una expresión aritmética dada en notación infija a una expresión en notación postfija.

Se van a considerar expresiones aritméticas bien formadas que tengan variables (de la 'a' a la 'z'), operadores binarios (+, -, *, /, ~) y paréntesis terminadas por la marca final '='.

El proceso de convertir acepta una *expresión infija* como entrada y produce una *expresión postfija* como salida.

PILA
Aplicación: Convertir infija a postfija

Idea: utilizar una pila para almacenar los operadores a medida que son encontrados para más tarde desapilar estos operadores de acuerdo a su precedencia.

TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

a + b = → a b + =

Pila:

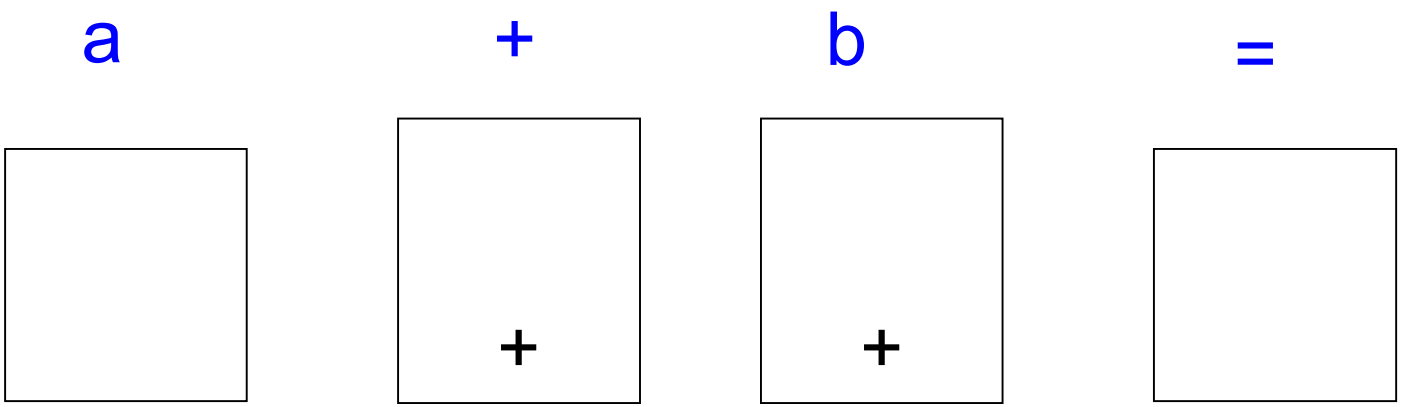


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

~a+b = → a ~ b + =

Pila:

~ a + b =

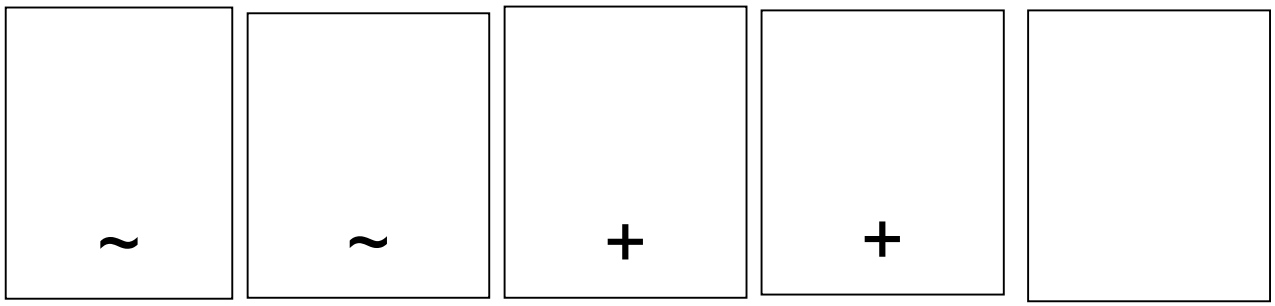


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

a - b + c = → a b - c + =

Pila:

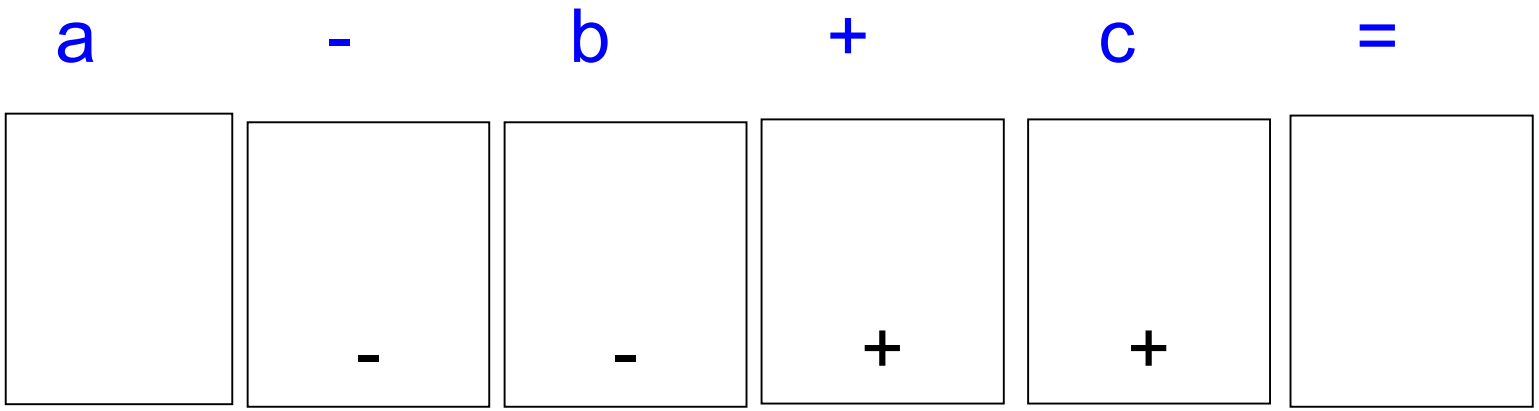


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

a * b + c = → a b * c + =

Pila:

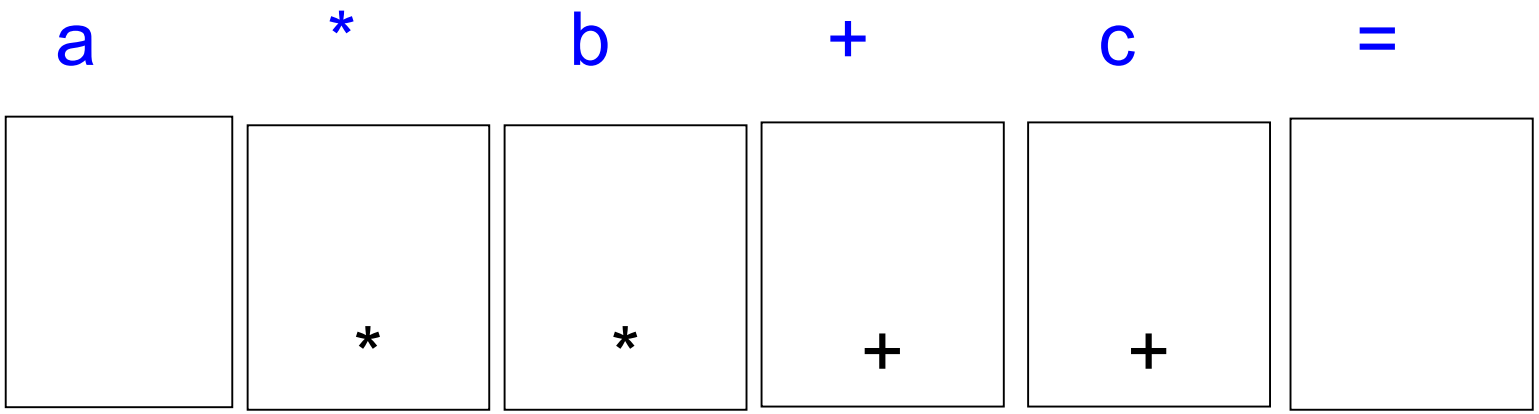


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

a + b * c = → a b c * + =

Pila:

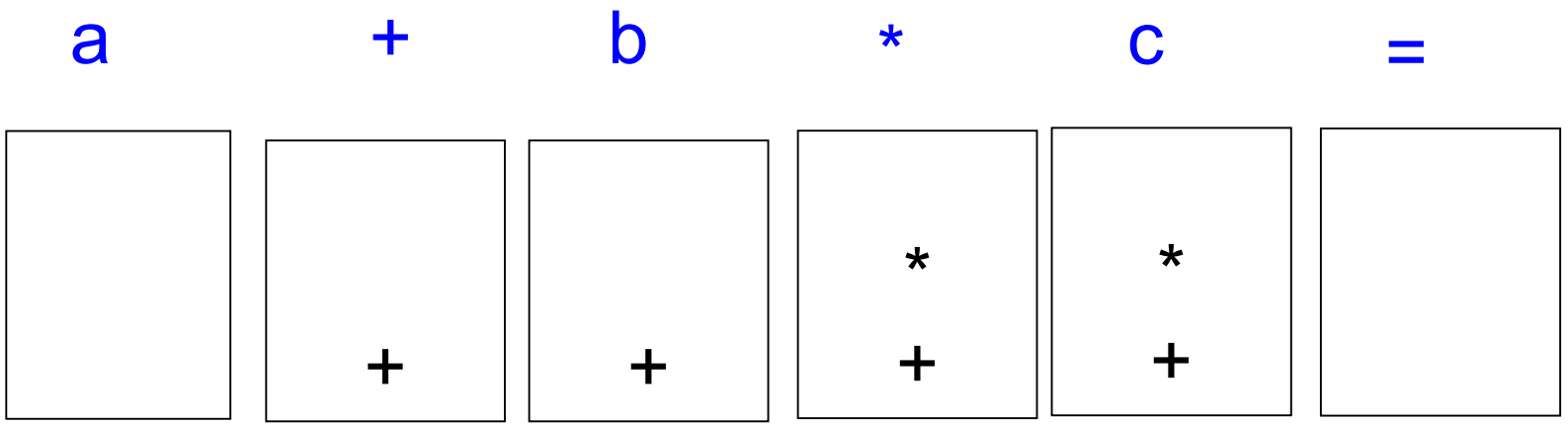


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

(a + b) * c = → a b + c * =

Pila:

(a + b) * c =

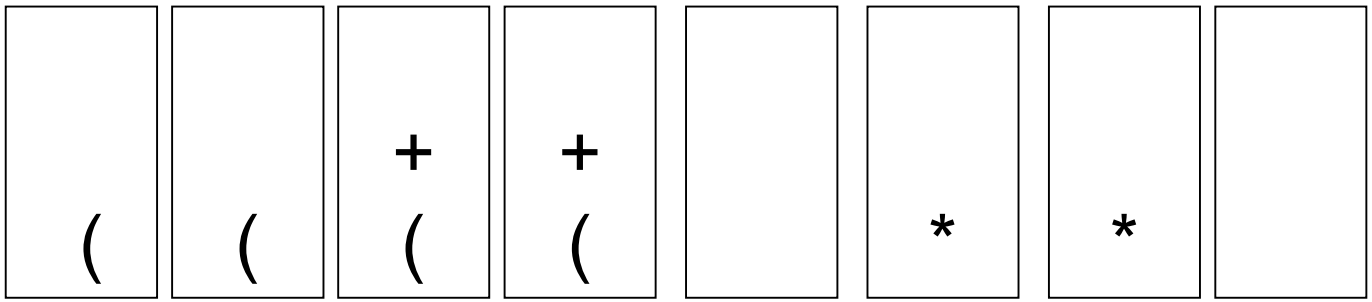


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA

Aplicación: Convertir infija a postfija

(a+ (~b))*c = → a b ~ + c * =

Pila:

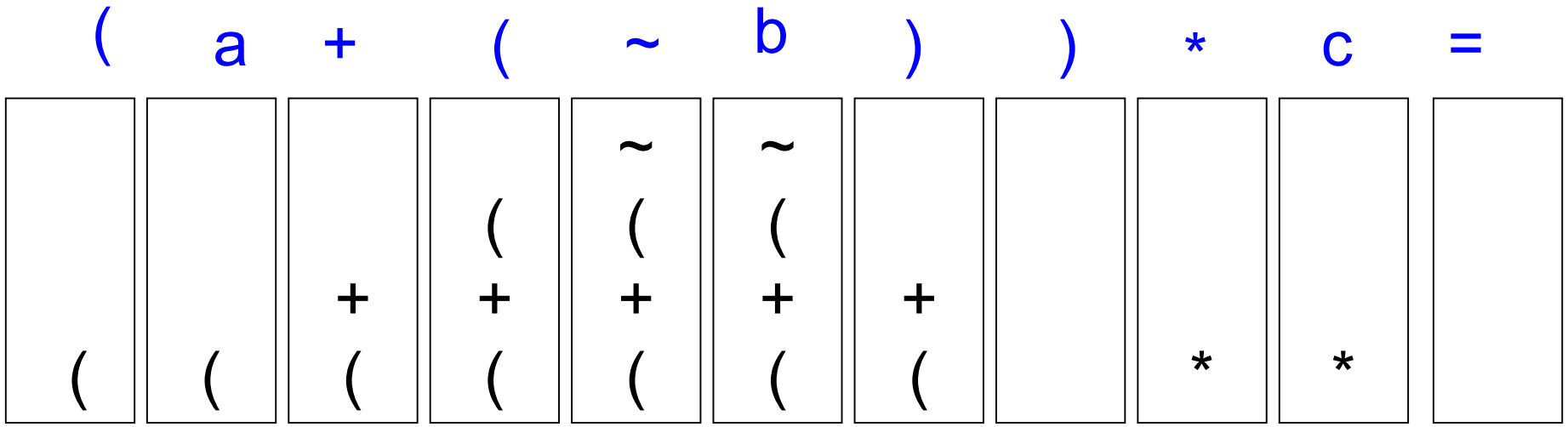


TABLA DE PRIORIDADES

OPERADOR	+	-	*	/	~	(	)	=
PRIORIDAD	1	1	2	2	3	0	-	0

PILA
Aplicación: Evaluar postfija

Escribir un algoritmo en términos del **ADT PILA** para **EVALUAR** una expresión aritmética en notación postfija.

Idea: utilizar una pila para almacenar los operandos a medida que son encontrados, cuando se encuentra un operador se sacan los operandos de la pila, se opera y el resultado vuelve a la pila.

PILA
Aplicación: Evaluar postfija

2 + 3 * 4 = → 2 3 4 * + = 14

Pila:

2	3	4	*	+	=
2	3 2	4 3 2	12 2	14	

PILA

Aplicación: Convertir infija a postfija

Forma infija:

9	+	2	*	(	5	*	4	-	6	)	/	7	-	(	8	+	3	*	1	)	=

tope

	+	+	* +	(* +	(* +	* (* +	* (* +	- (* +	- (* +	* +	/ +	/ +	-	(-	(-	+ (-	+ (-	* + (-	* + (-	-	

fondo

Contenido de la pila *después de procesar cada carácter* para pasar de infija a postfija

Forma postfija:

9	2	5	4	*	6	-	*	7	/	+	8	3	1	*	+	-	=

PILA

Aplicación: Evaluar postfija

Forma postfija:

9	2	5	4	*	6	-	*	7	/	+	8	3	1	*	+	-	=

Evaluar la posfija:

Contenido de la pila *después de evaluar cada carácter* de la postfija*:*

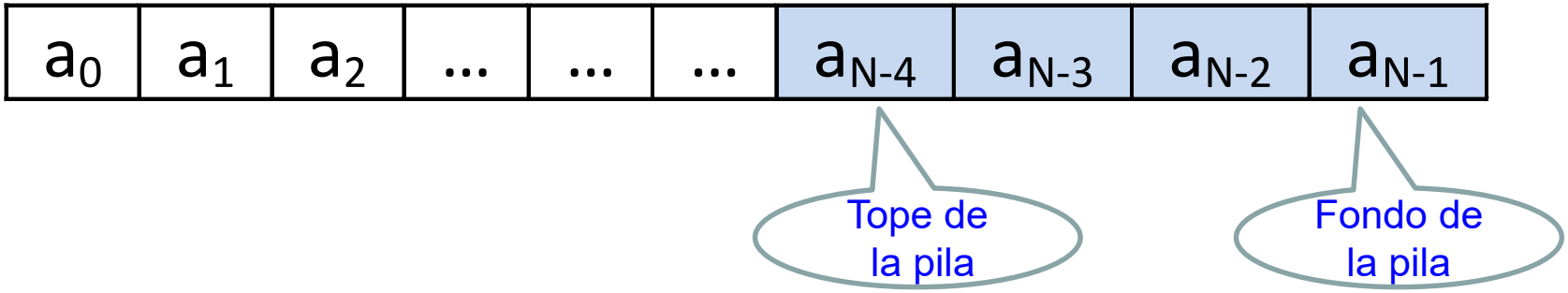
9	2 9	5 2 9	4 5 2 9	20 2 9	6 20 2 9	14 2 9	28 9	7 28 9	4 9	13	8 13	3 8 13	1 3 8 13	3 8 13	11 13	2	

PILA
Implementación

- Para la representación de las pilas se requiere el uso de otras estructuras de datos como ser:

  – Arreglos (arrays)
  – Listas enlazadas

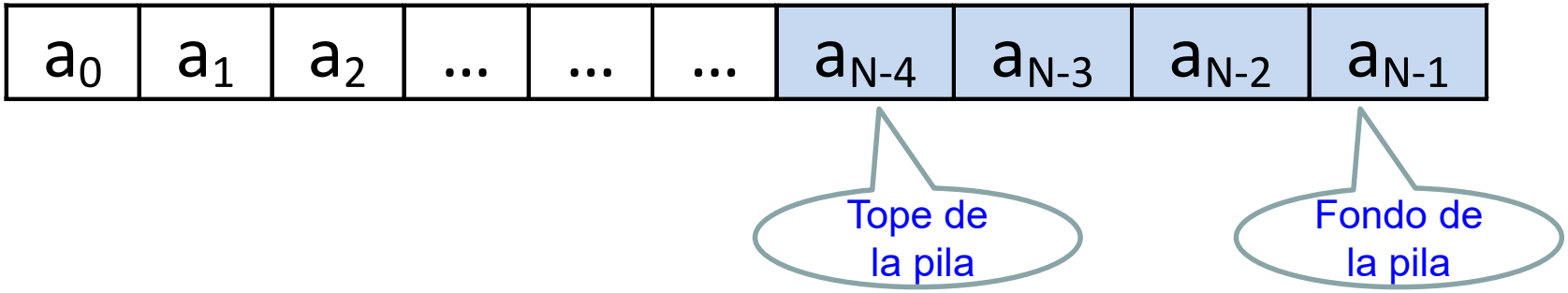
PILA
Implementación con Arreglo en C




```
// PRIVADO
#define  N 100
#define indefinido -9999
typedef int item;
typedef struct {     int tope;
            item elementos [N];}     tipopila;
```

```
//PUBLICO
void pilavacia ( tipopila* );
int espilavacia ( tipopila );
item top ( tipopila );
void pop ( tipopila* );
void push ( tipopila*, item );
```

PILA
Implementación con Arreglo en C




```
// PRIVADO
#define  N 100
#define indefinido -9999
typedef int item;
typedef struct {     int tope;
            item elementos [N];}     tipopila;
```

```
//PUBLICO
void pilavacia ( tipopila* );
int espilavacia ( tipopila );
item top ( tipopila );
void pop ( tipopila* );
void push ( tipopila*, item );
```

PILA
Implementación con Arreglo en C

```
void pilavacia( tipopila *pila)
  { pila->tope = N; }

int espilavacia(tipopila pila)
  { return(pila.tope == N); }

item top ( tipopila pila)
  { if( espilavacia(pila) )
        return (indefinido);
    else
        return(pila.elementos[pila.tope]); }

void pop( tipopila *pila)
  { if ( ! espilavacia(*pila) )   pila->tope++;  }

void push (tipopila *pila, item x)
  { if( pila->tope != 0 )
        pila->elementos[--pila->tope] = x;   }
```

// NOTA: todas las operaciones son O(1) en esta implementación

PILA
Programa de prueba en C

```
#include <iostream>
using namespace std;

int main()
{    tipopila PP;
     pilavacia(&PP);
     cout<<endl<<"probando pila con arreglo en C" <<endl;
     cout<<"ESTA vacia: " << espilavacia(PP)<<endl;
     for (int i=8; i<16; i++)
          {push( &PP,i);
          cout<<"push de " << i <<endl;    }
     cout<<"ESTA vacia: " << espilavacia(PP)<<endl;
     cout<<"-------Imprimiendo pila-------"<<endl;
     while (!espilavacia(PP))
          {cout<<"tope: "<<top(PP)<<endl;
           pop(&PP);}
cout<<"ESTA vacia: "<<espilavacia(PP)<<endl;
cout<<"tope: "<<top(PP)<<endl;
return 0;
}
```

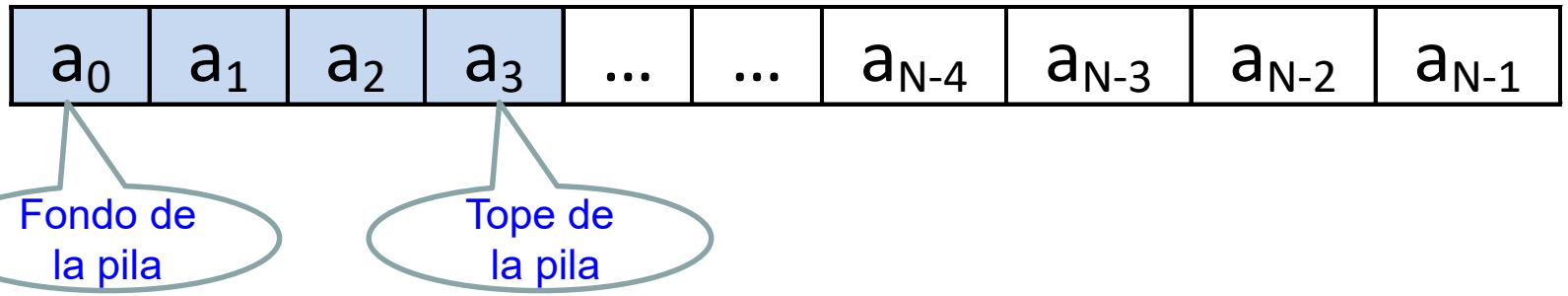
PILA

Salida del Programa de prueba

```
probando pila con arreglo en C
ESTA vacia: 1
push de 8
push de 9
push de 10
push de 11
push de 12
push de 13
push de 14
push de 15
ESTA vacia: 0
-------Imprimiendo pila-------
tope: 15
tope: 14
tope: 13
tope: 12
tope: 11
tope: 10
tope: 9
tope: 8
ESTA vacia: 1
tope: -9999
```

PILA

Otra implementación con arreglo en C

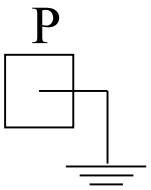



```
// PRIVADO
#define  N 100
#define indefinido -9999
typedef int item;
typedef struct {    int tope;
            item elementos [N];}    tipopila;
//PUBLICO
void pilavacia ( tipopila* );
int espilavacia ( tipopila );
item top ( tipopila );
void pop ( tipopila* );
void push ( tipopila*, item );
```

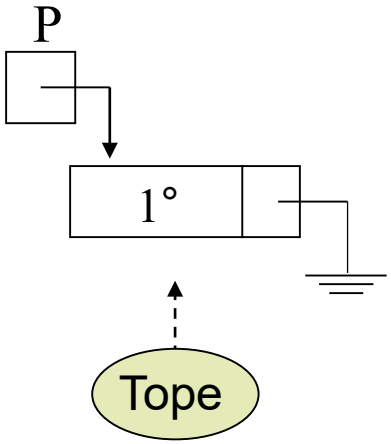
PILA

Implementación con Listas enlazadas en C

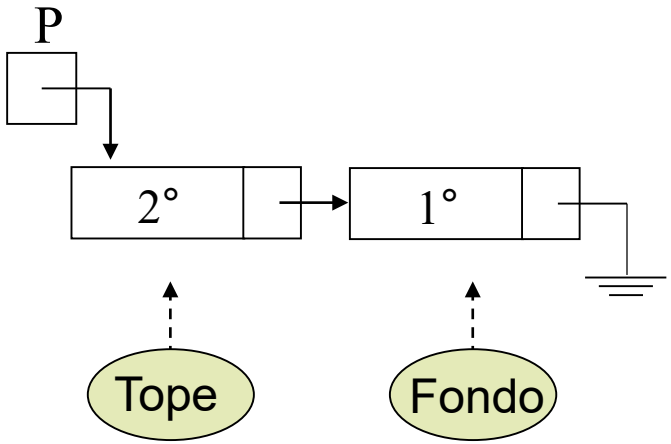
PILAVACÍA



PILA CON UN ELEMENTO



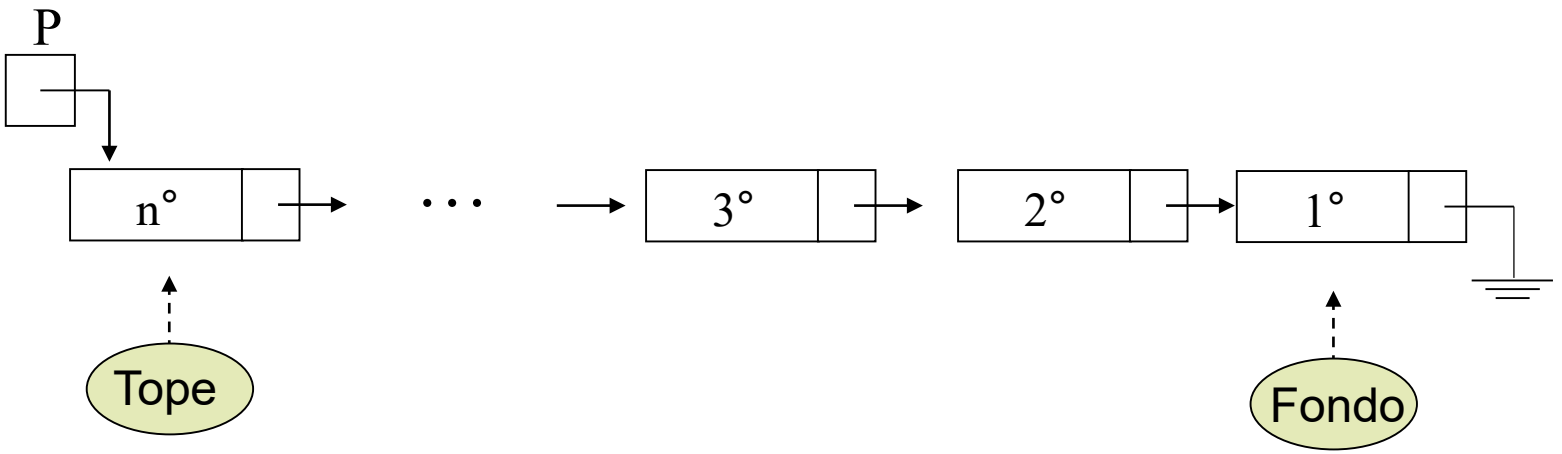
PILA CON DOS ELEMENTOS



PILA

Implementación con Listas enlazadas en C

PILA CON n ELEMENTOS



PILA

Implementación con Listas enlazadas en C

```
// PRIVADO
typedef int item;
const item indefinido=-9999;
struct nodo {
              item dato;
              struct nodo* sig;
             };
typedef struct nodo* tipopila;
```

// NOTA: todas las operaciones son O(1) en esta implementación