Unidad 1

Fundamentos de Programación Competitiva

Logro de sesión

Al finalizar la sesión, el estudiante comprenderá estructuras de datos

Semana 3

Estructuras de datos

Contenido:

- Estructura de datos
- Estructura estáticas (Arreglos y Matrices)
- Estructuras dinámicas (Pilas y Colas)

Estructura de datos

- Una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación.
- Un dato elemental es la mínima información que se tiene en un sistema.
- Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación.

Estructuras Estáticas

Las estructuras estáticas son aquellas que el tamaño de las mismas no puede ser modificadas en la ejecución del algoritmo.

Se divide en:

- Estructuras Estáticas Simples
- Estructuras Estáticas Compuestas

Estructuras Estáticas

- Estructuras Estáticas Simples

  Estas estructuras de datos son también llamadas Primitivas.

  - ✓ Booleanas - Bool (0 -1, false – true)
  - ✓ Carácter - Char
  - ✓ Enteras - Int
  - ✓ Flotante – Float
  - ✓ Largo – Long long
  - ✓ Double

Estructuras Estáticas Simples

Estructuras Estáticas Simples

Ejemplos:

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

int main()
{
   int x,y;
   char c;
   bool valor;
      long long numero;
      float dec;
```

```
x= 10; y =20;
C = 'E';
valor = 0;
numero = 123456985632;
dec = 142,15;

cout<<"\nValor  Entero:  "<<x<<"  -
"<<y;
cout<<"\nValor Char: "<<c;
cout<<"\nValor Bool: "<<valor;
cout<<"\nValor Long long: "<<numero;
cout<<"\nValor Float: "<<dec;
return 0;
}
```

Estructuras Estáticas Compuestas

- *Estructuras* Estáticas *Compuestas*

  Esta estructura almacena varios elementos del mismo tipo en forma lineal.

Vector [5], vector de tamaño 5.

15	2	36	4	28

Matriz [3][4], matriz de tamaño 3x4.

10	36	4	19
18	96	-3	9
68	13	8	3

Estructuras Dinámicas

Las estructuras dinámicas lineales son: Pilas y Colas.

Pilas

- Una pila es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos.

Pilas – Problema1

- Implementar la posibilidad de deshacer el último cambio realizado en una aplicación ofimática (Word, Excel, etc.)
  - Crtl-Z
- ¿Hace falta almacenar información para realizar esta tarea?
- ¿Cómo se debe organizar el almacenamiento?
  - ¿Dónde se ubica un dato cuando se almacena?
  - ¿Qué dato se debe extraer?

Pilas (LIFO)

Estructura

Pila (Valor)

Valor constituye el dominio sobre el que se define el tipo de datos. Hace referencia a los datos que se almacenan en la pila.

Operaciones

CrearPila () ⟶ Pila
Apilar (Pila , Valor) → Pila
Desapilar (Pila) → Pila
CimaPila (Pila) → Valor
PilaVacia (Pila) → Lógico

Axiomas

Para todo s Є Pila, x Є Valor se cumple que:

PilaVacia (CrearPila ()) → cierto
PilaVacia (Apilar (s, x)) → falso
Desapilar (CrearPila ()) → error
Desapilar (Apilar (s, x)) → s
CimaPila (CrearPila ()) → error
CimaPila (Apilar (s, x)) → x

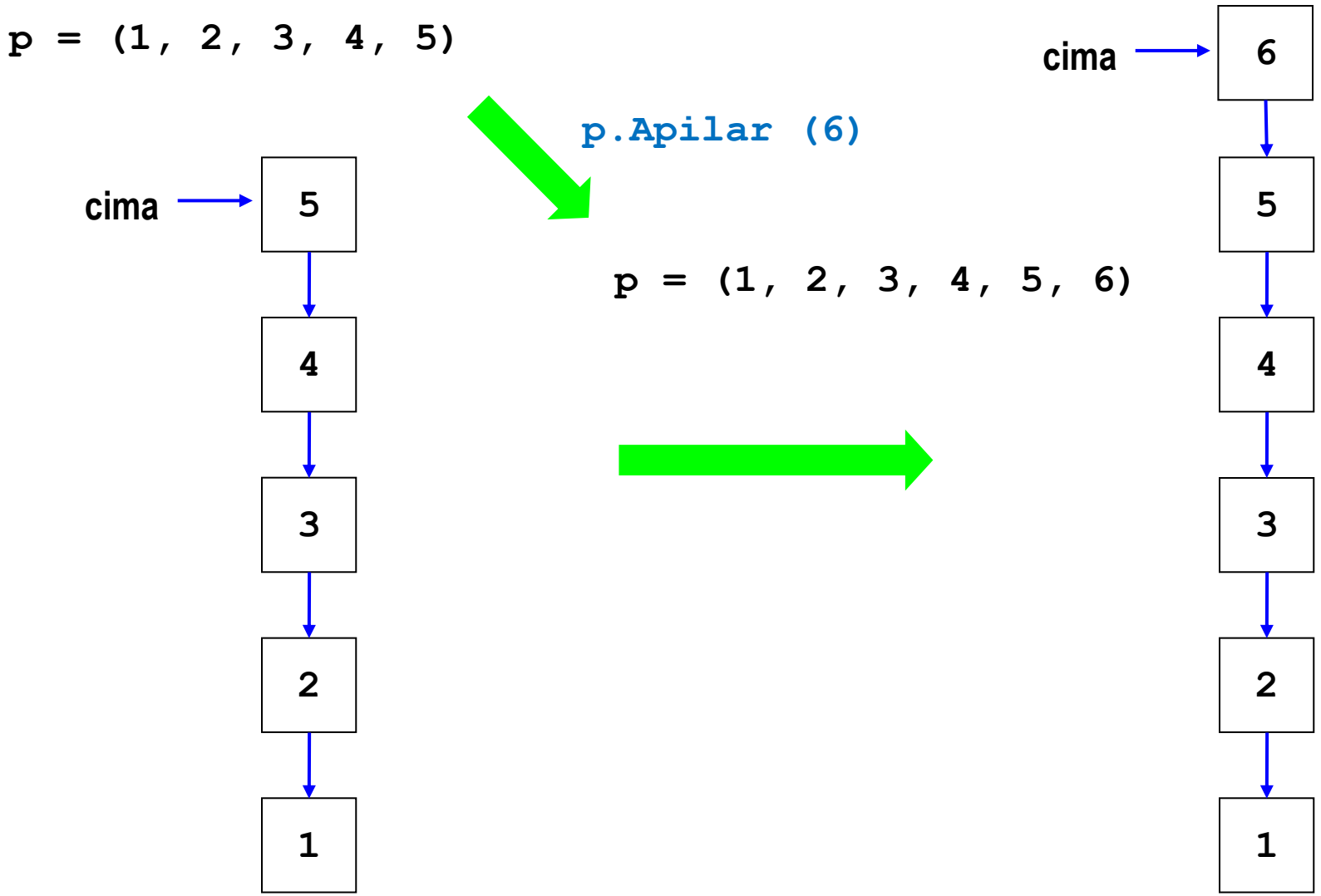
Stack (STL)

- La librería estándar incluye un contenedor `stack` (pila).

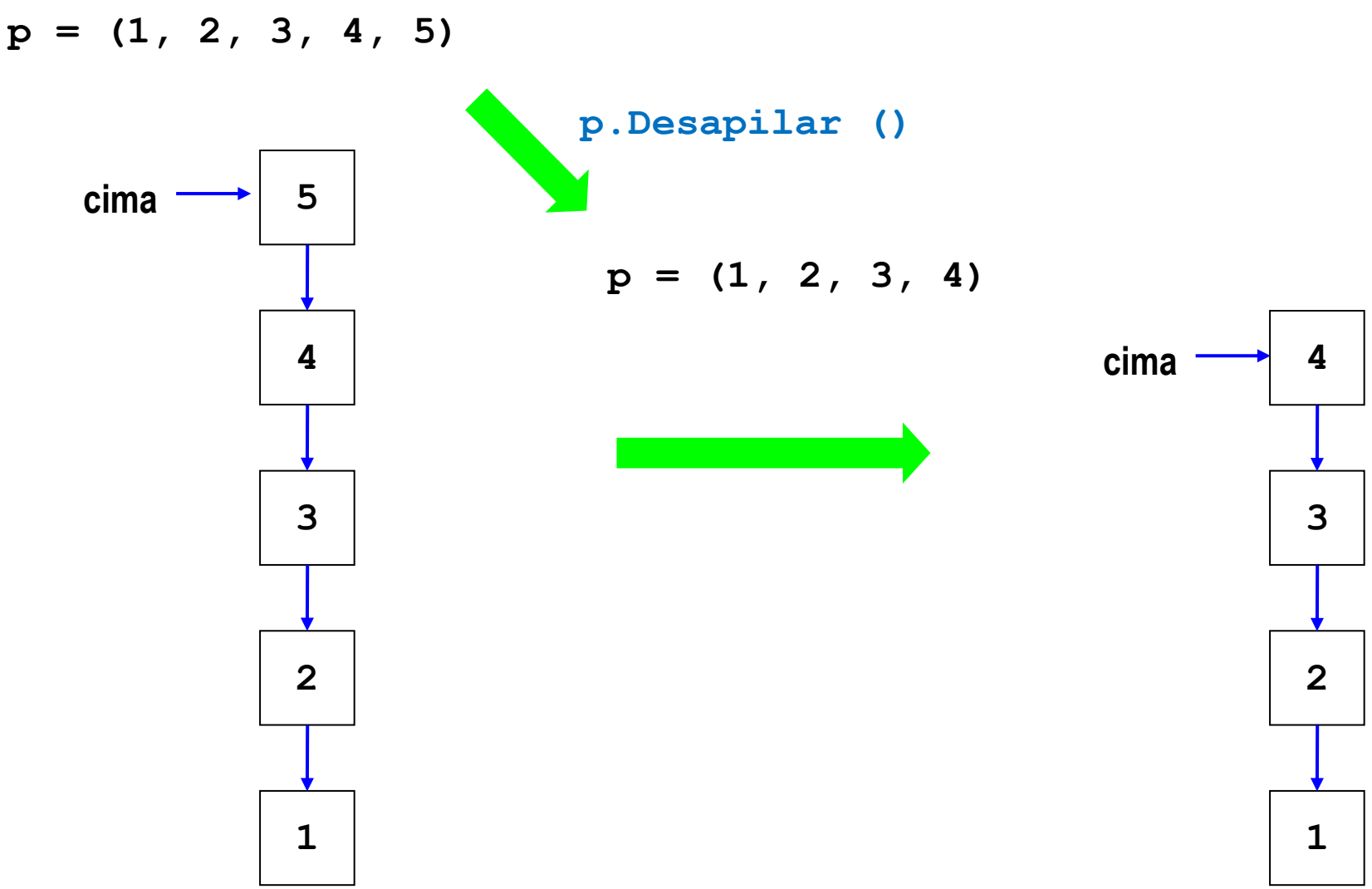
- Declaración de un objeto pila:
  - **`stack<Valor> pila; //#include <stack>`**
  - donde **`Valor`** es el tipo de los elementos contenidos en la pila.

- Operaciones de stack:
  - `bool` **`empty`**`();` //true si la pila está vacía = *PilaVacia*
  - `Valor&` **`top`**`();` //Referencia a la Cima de la pila = *CimaPila*
  - `void` **`push`**`(x);` //*Apilar* x en la pila
  - `void` **`pop`**`();` //*Desapilar*
  - `unsigned int` **`size`**`();` //número de elementos almacenados en la pila

Apilar (push)



Desapilar (pop)



Ejemplo

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

int main ()
{
  stack<int> pila; //pila de enteros
  int suma = 0;

  for (int i=1;i<=10;i++) pila.push(i); //apilar de 1 a 10

  cout << "tamanyo: " << pila.size() << endl;

  while (!pila.empty()) //mientras la pila no esté vacía
  {
     suma += pila.top(); //sumar el valor de la cima
     pila.pop(); //desapilar 1 elemento
  }

  cout << "total: " << suma << endl;

  return 0;
}
```

```
tamanyo: 10
total: 55
```

Colas

- Una cola es una estructura de datos en la que el modo de acceso a sus elementos es de tipo FIFO (del inglés First In First Out, primero en entrar, primero en salir) que permite almacenar y recuperar datos. Así, los elementos sólo se pueden eliminar por el principio y sólo se pueden añadir por el final de la cola.

Colas

Estructura

Cola (Valor)

Operaciones

CrearCola () → Cola
Encolar (Cola , Valor) → Cola
Desencolar (Cola) → Cola
PrimeroCola (Cola) → Valor
ColaVacia (Cola) → Lógico

Valor constituye el dominio sobre el que se define el tipo de datos.
Hace referencia a los datos que se almacenan en la cola. En cada aplicación el tipo cambia.

Axiomas

Para todo q ∈ Cola, x ∈ Valor se cumple que:

ColaVacia(CrearCola ()) → cierto
ColaVacia(Encolar (q, x)) → falso
Desencolar(CrearCola ()) → error

Desencolar(Encolar (q, x)) →

$$\begin{cases} \text{CrearCola() si ColaVacia(q)=cierto} \\ \text{Encolar(Desencolar (q), x) sino} \end{cases}$$

PrimeroCola(CrearCola ()) → error

PrimeroCola(Encolar (q, x)) →

$$\begin{cases} \text{x si ColaVacia(q)= cierto} \\ \text{PrimeroCola(q) sino} \end{cases}$$

queue

- La librería estándar incluye un contenedor `queue` (cola).

- Declaración de un objeto pila:

  `queue<Valor> cola; //#include <queue>`

  donde **`Valor`** es el tipo de los elementos contenidos en la cola.

- Operaciones de `queue`:

```
bool empty(); //true si la cola está vacía = ColaVacia
Valor& front(); //Referencia al primero de la cola = PrimeroCola
Valor& back(); //Referencia al último de la cola
void push(x); //Encolar x
void pop(); //Desencolar
unsigned int size(); //número de elementos en la cola
```

Ejemplo

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

int main ()
{
  queue<int> cola; //cola de enteros
  int suma = 0;

  for (int i=1;i<=10;i++) cola.push(i); //encolar de 1 a 10

  cout << "tamanyo: " << cola.size() << endl;

  while (!cola.empty()) //mientras la cola no esté vacía
  {
     suma += cola.front(); //sumar el valor del primero de la cola
     cola.pop(); //desencolar 1 elemento
  }

  cout << "total: " << suma << endl;

  return 0;
}
```

```
tamanyo: 10
total: 55
```

Muchas Gracias!!!