Programación 2

Introducción al curso

Objetivos del curso

- **Presentar y analizar estructuras de datos y algoritmos que forman la base para la resolución de muchos problemas relevantes en computación.**
- **Introducir y aplicar nociones de análisis de algoritmos.**
- **Aprender a implementar sistemas eficientes de porte mediano, considerando nociones de abstracción, especificación, implementación y uso de módulos.**

Tópicos

- Introducción
  - Nociones Generales.
  - Abstracción y Modularización.

- Introducción al Análisis de Algoritmos
  - Eficiencia en espacio de almacenamiento y tiempo de ejecución.
  - Tiempos de ejecución y órdenes. Definiciones y algunas propiedades.
  - Cálculo de tiempos de ejecución y órdenes para programas iterativos.

Tópicos

- Recursión y Análisis de Algoritmos Recursivos
  - Definición de tipos de datos inductivos. Inducción y esquemas de recursión asociados.
  - Programación recursiva: tipos y aplicaciones.
  - Recursión en un sistema computacional: el stack de ejecuciones.
  - Introducción al Análisis de Algoritmos Recursivos

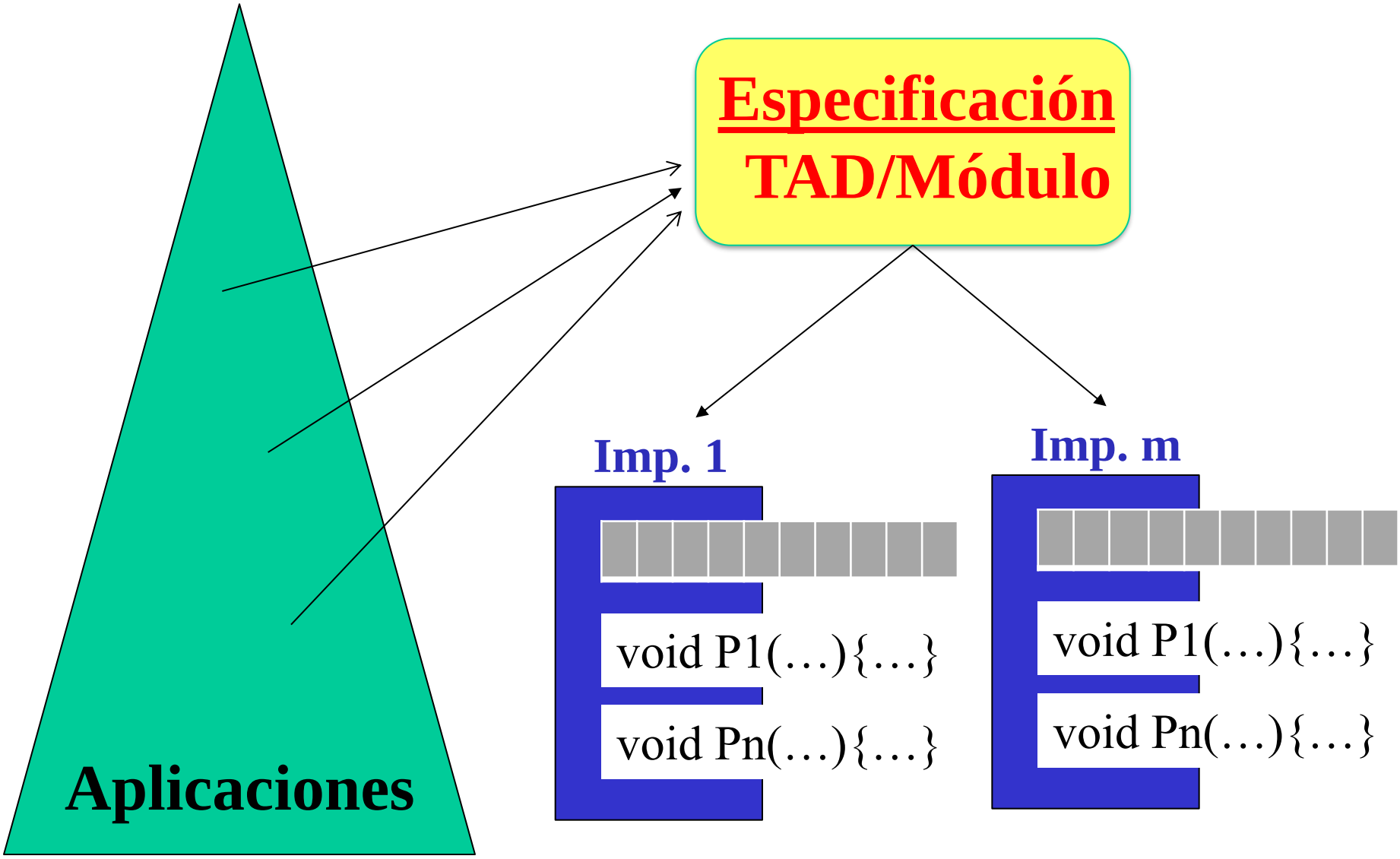
- Estructuras Dinámicas
  - Estructuras estáticas y estructuras dinámicas. Punteros y manejo de memoria dinámica.
  - Listas de memoria dinámica. Algoritmos sobre listas.
  - Estructuras arborescentes de memoria dinámica. Algoritmos sobre estructuras arborescentes.
  - Aplicación de conceptos de análisis de algoritmos sobre estructuras dinámicas.

Tópicos

- Tipos Abstractos de Datos (TADs)
  - Abstracción procedural y abstracción de datos.
  - Especificación de TADs: uso de pre y post condiciones.
  - Implementación de TADs usando estructuras de datos de variada complejidad, incluyendo también estructuras múltiples (multiestructuras).
  - Uso de TADs.
  - Análisis de las ventajas de la programación con TADs.

- TADs Fundamentales
  - Especificación e implementaciones de TADs fundamentales. Por ejemplo: Listas, Pilas, Colas, Conjuntos, Funciones Parciales (Tablas, Mappings), Multiconjuntos y Colas de Prioridad. Variantes.
  - Aplicaciones (uso de TADs).
  - Introducción al diseño de TADs.

Sobre TADs, Abstracción y Modularización



Materiales del curso

Sitio oficial: EVA de Programación 2

Organización por temas:

- Teóricos
  - Guías.
  - Videos en OpenFING (se renovarán durante el primer semestre de 2023) + foros.
- Prácticos
  - Letras.
  - Algunas soluciones.
  - Grupos presenciales/vitrtual: Ver días, horarios y salones en el sitio EVA. Cada grupo tiene dos instancias semanales de 2hs (4hs en total) + foros.
- Laboratorio
  - Letras, materiales, foros, entregas y reentregas; todo por EVA.

Sobre el curso y su evaluación

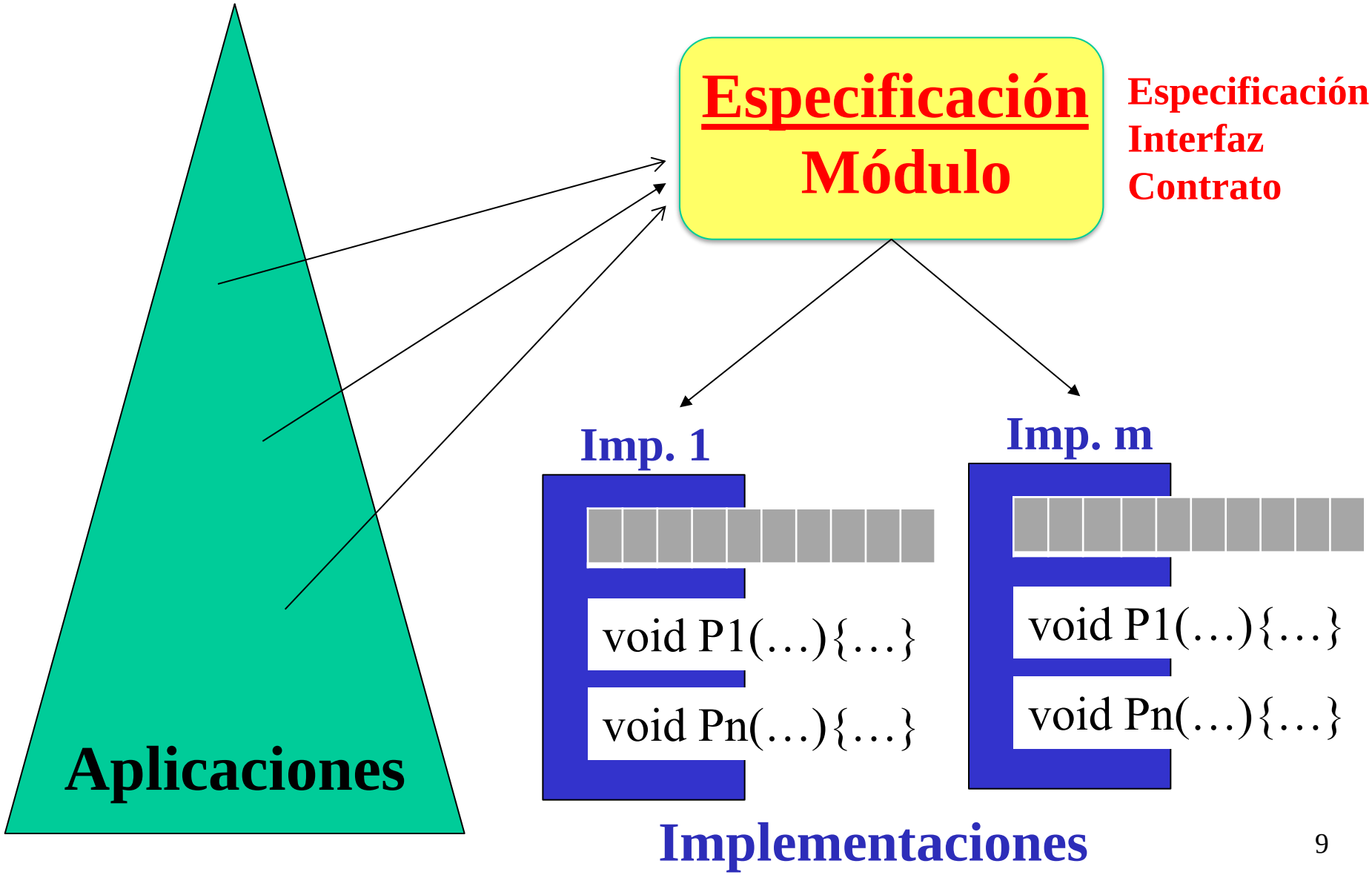
Ver en el Foro Novedades del curso en el EVA el Mensaje de bienvenida con información del curso.

EVALUACIÓN:

- **Parciales: 92 (37 + 55)**
- **Laboratorios: 8**
- **Información sobre el procedimiento de evaluación 2023:** https://eva.fing.edu.uy/mod/page/view.php?id=73810

Tener presente los reglamentos y el programa del curso disponibles en el EVA.

Sobre
Abstracción y Modularización

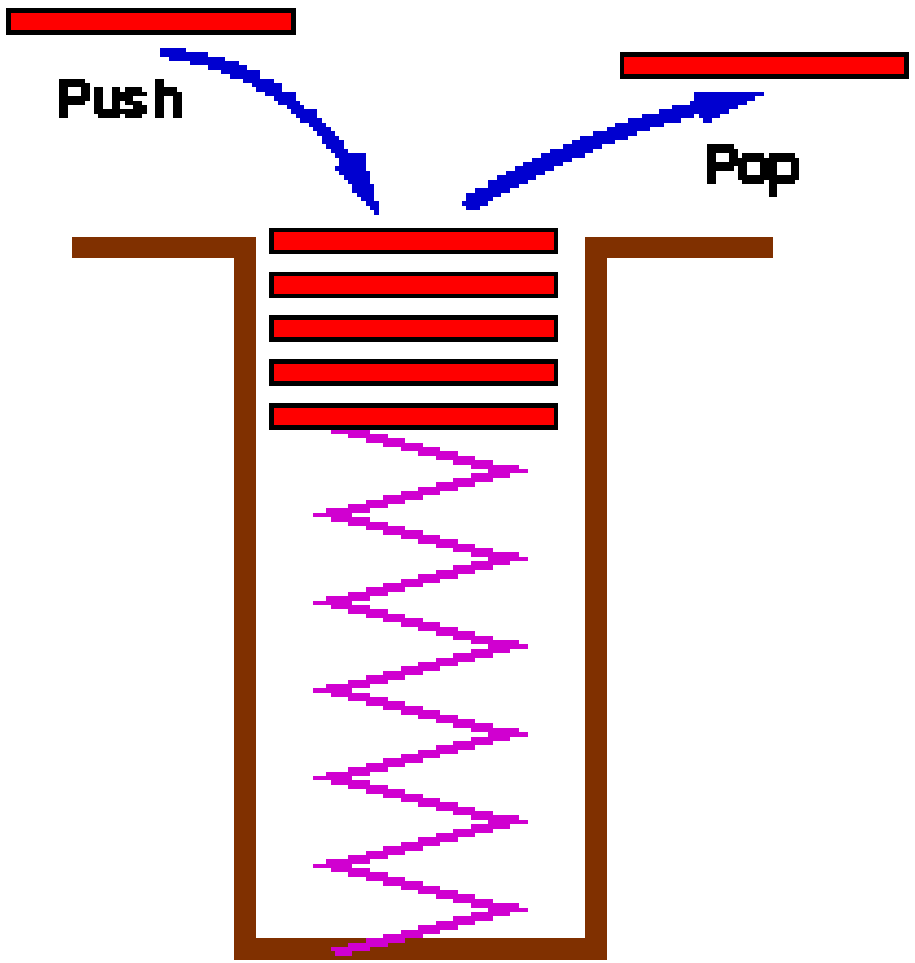


Ejemplo:

PILA (STACK)

Definición

- Política ***LIFO*** (last-in-first-out).



Operaciones

Considere el Módulo Stack (Pila) con (entre otras) las siguientes operaciones para su *especificación / interfaz / contrato*:

- **`Empty`** (**`crearPila`**), que construye un stack vacío;
- **`Push`** (**`apilar`**), que inserta un elemento en el tope del stack;
- **`Top`** (**`cima`**), que retorna el elemento que se encuentra en el tope del stack (si no es vacío);
- **`Pop`** (**`desapilar`**), que remueve el elemento que se encuentra al tope (en la cima) del stack;
- **`IsEmpty`** (**`esVaciaPila`**), que testea si el stack es vacío o no;
- **`IsFull`** (**`esLlenaPila`**), si se trata de un stack acotado, que testea si el stack está lleno.

Especificación en C/C++ del módulo Pila acotada

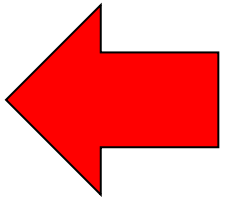
```
#ifndef _PILA_H
#define _PILA_H

struct RepresentacionPila;
typedef RepresentacionPila * Pila;

// OPERACIONES

Pila crearPila (int cota);
// Devuelve la pila vacía que podrá contener hasta cota elementos.

void apilar (int i, Pila &p);
/* Si !esLlenaPila(p) inserta i en la cima de p,
   en otro caso no hace nada. */

int cima (Pila p);
/* Devuelve la cima de p.
   Precondicion: ! esVaciaPila(p). */

void desapilar (Pila &p);
/* Remueve la cima de p.
   Precondicion: ! esVaciaPila(p). */
```

Especificación en C/C++ del módulo Pila acotada

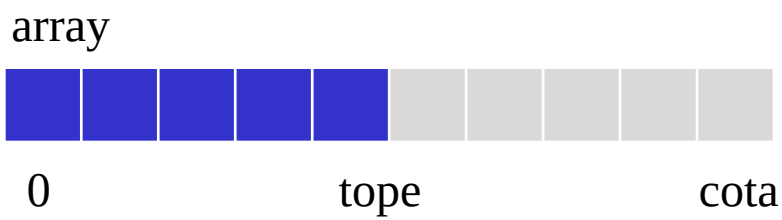
```
bool esVaciaPila (Pila p);
/* Devuelve true si y sólo si p es vacia. */

bool esLlenaPila (Pila p);
/* Devuelve 'true' si y sólo si p tiene cota elementos, donde cota
es el valor del parámetro pasado en crearPila. */

void destruirPila (Pila &p);
/* Libera toda la memoria ocupada por p. */


#endif /* _PILA_H */
```

Implementación en C/C++ de una Pila acotada de enteros usando un array con tope

```
#include <stddef.h>
#include <assert.h>

#include "pila.h"

struct RepresentacionPila{
        int* array;
        int tope;
        int cota;
};

// OPERACIONES
Pila crearPila (int cota) {
        Pila p = new
           RepresentacionPila;
        p->tope = 0;
        p->array = new int cota];
        p->cota = cota;
        return p;
}
```



Veremos implementaciones de Módulos/TADs más adelante



Ejemplo de uso del módulo Pila

```
#include <stdio.h>
#include "pila.h"
void main{
      Pila p;
      p = crearPila(100);
      for (int i = 1, i <= 50, i++)
            apilar(i, p);
      while (!esVaciaPila(p)){
                  printf("%d\n", cima(p))
                  desapilar(p);
      }
      destruirPila(p);
}
```

Notar que este código no depende de una implementación de la Pila

Algunas ventajas

- Adecuado para sistemas no triviales; descomposición de la complejidad en módulos.
- Separación entre especificación e implementación. Esto hace al sistema:
  - más legible
  - más fácil de mantener
  - más fácil de verificar y probar que es correcto. Robustez.
  - más fácil de reusar
  - más extensible
  - lo independiza en cierta manera de las distintas implementaciones (eficiencia).

Conclusiones

- La genericidad es una característica que buscaremos desarrollar y explotar para fomentar el reuso.
- El uso de estos módulos permite una rápida prototipación de sistemas, dejando de lado detalles de eficiencia para un etapa posterior.
- …..