Programación 2

La Previa 1

Introducción al curso

Objetivos del curso

- **Presentar y analizar las estructuras de datos y algoritmos que forman la base para la resolución de problemas en computación;**
- **Introducir nociones de análisis de algoritmos;**
- **Aprender a implementar sistemas (eficientes) de porte mediano.**

Materiales del curso

Sitio oficial: EVA de Programación 2

Organización por temas:

- Teóricos
  - Guías
  - Videos en OpenFING
  - Resumen y consultas + foros
- Prácticos
  - Letras
  - Algunas soluciones escritas y videos
  - Varios grupos: presenciales y uno remoto (por Zoom). Cada grupo tiene dos instancias semanales de 1:30hs (3hs en total) + foros
- Laboratorio
  - Letras, materiales, foros, entregas y reentregas; todo por EVA

Materiales del curso

- Guías de clases teóricas (incluyen ejercicios)

- Libros (básicos)
  - Data structures and Algorithm analysis in C y C++
    Mark Allen Weiss
  - Estructuras de Datos y Algoritmos
    A. Aho, J. E. Hopcroft & J. D. Ullman
  - Cómo Programar en C/C++
    H.M. Deitel & P.J. Deitel
    O la versión 2: Cómo programar en C++

Sobre el curso y su evaluación

Ver en Novedades de EVA:

- Prog2 » Foros » Foro de Novedades » Bienvenida

Tener presente los reglamentos y el programa del curso.

EVALUACIÓN:

- **Parciales: 92 (37 + 55)**
- **Laboratorios: 8**
- **Más información en EVA:** https://eva.fing.edu.uy/mod/page/view.php?id=73810

Tópicos

- Introducción
  - Nociones Generales.
  - Abstracción en programación: particiones-refinamientos. Abstracción procedural e introducción a la abstracción de datos. Compilación separada de módulos.

- Análisis de Algoritmos
  - Introducción al análisis de algoritmos. Eficiencia en espacio de almacenamiento y tiempo de ejecución. Tiempo de ejecución, orden del peor caso y caso promedio. Propiedades. Cálculo de tiempo de ejecución para programas iterativos e introducción al cálculo de tiempo de ejecución para programas recursivos.

Tópicos

- Inducción y Recursión
  - Recursión en un sistema computacional. El rol del stack de ejecuciones. Cuándo conviene usar recursión? Análisis de iteración vs recursión.
  - Definición de tipos de datos inductivos. Esquemas de inducción y recursión asociados.
  - Programación recursiva: tipos y aplicaciones. Programación recursiva con precondiciones.

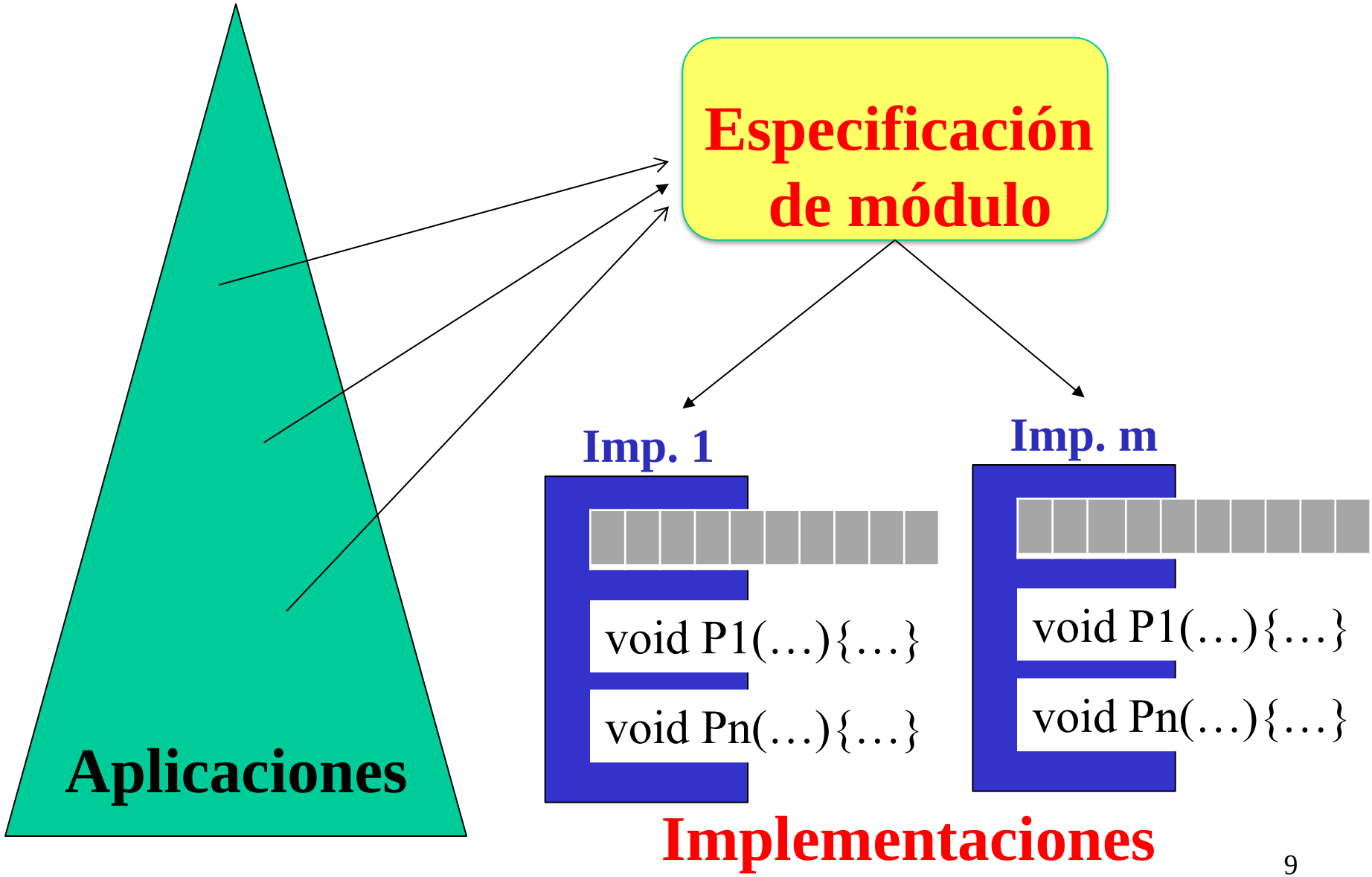
- Estructuras Dinámicas
  - Estructuras estáticas y estructuras dinámicas. Punteros y manejo de memoria dinámica. Definición de listas de memoria dinámica. Algoritmos sobre listas. Definición de estructuras arborescentes de memoria dinámica. Algoritmos sobre estructuras arborescentes.

Tópicos

- Introducción a Tipos Abstractos de Datos (TADs)
  - Abstracción procedural y abstracción de datos.
  - Especificación de TADs: uso de pre y post condiciones. Implementación de TADs. Uso de TADs. Análisis de las ventajas de la programación con TADs.

- TADs Fundamentales
  - Especificación e implementaciones eficientes de TADs fundamentales. Por ejemplo: Listas, Pilas, Colas, Conjuntos, Diccionarios y Funciones parciales. Variantes. Aplicaciones.

Sobre Módulos



Semanas del 5/8 y 12/8

TEMA 0 - Introducción y Programas en C*

TEMA 1 - Abstracción y Modularización

Veamos algo del práctico de la semana próxima…

El problema 1 (analizamos las distintas partes del ejercicio 1 del práctico de modularización).

Ordenación

Ver algoritmos de ordenación de Programación 1

Ejemplos:

- Selection sort: Busca el mínimo elemento, lo pone al inicio y luego aplica la misma estrategia a los restantes elementos (todos salvo el primero).

- Insertion sort: Recorre una secuencia e inserta de manera ordenada cada elemento en una nueva secuencia originalmente vacía.

Selection Sort

Consideremos un arreglo **lista** de largo n ([0 : n-1]), n>0

```
for (int i = 0 ; i < n-1 ; i++){
    int pos_min = i;
    for (int j = i + 1 ; j < n ; j++){
        if (lista[j] < lista[pos_min]){
            pos_min = j;
        }
    }
    intercambiar (lista, i, pos_min); /* intercambia de
    lista los elementos en las posiciones i y pos_min */
}
```

Selection Sort

Definiendo ahora una función auxiliar:

```
for (int i = 0 ; i < n-1 ; i++){
    int pos_min = pos_minimo (lista, i, n-1);
    intercambiar (lista, i, pos_min); // ¿Si i==pos_min?
}
```

/* Retorna la posición del mínimo de un arreglo *a* entre las posiciones *ini* y *fin*, con *ini* < *fin* */

```
int pos_minimo (int * a, int ini, int fin)
```

Selection Sort

/* Retorna la posición del mínimo de un arreglo *a* entre las posiciones *ini* y *fin*, con *ini* < *fin* */

```
int pos_minimo (int * a, int ini, int fin)
    int pos_min = ini;
    for (int i = ini + 1 ; i <= fin ; i++){
            if (a[i] < a[pos_min]){
                pos_min = i;
            }
    }
    return pos_min;
}
```

Un ejemplo

Consideremos un arreglo **lista** de largo n ([0 : n-1]), n>0

```
for (int i = 0 ; i < n-1 ; i++){
    for (int j = n-1 ; i < j ; j--){
            if (lista[j-1] > lista[j]){
                    intercambiar (lista, j-1, j);
            }
     }
}
```

También ordena. Es el *Bubble sort*.

¿Se parece al *Selection sort*?

La previa del 16/8

El Laboratorio:

- **Funcionamiento**
- **Lenguaje, materiales y herramientas**
- **Primeras tareas**