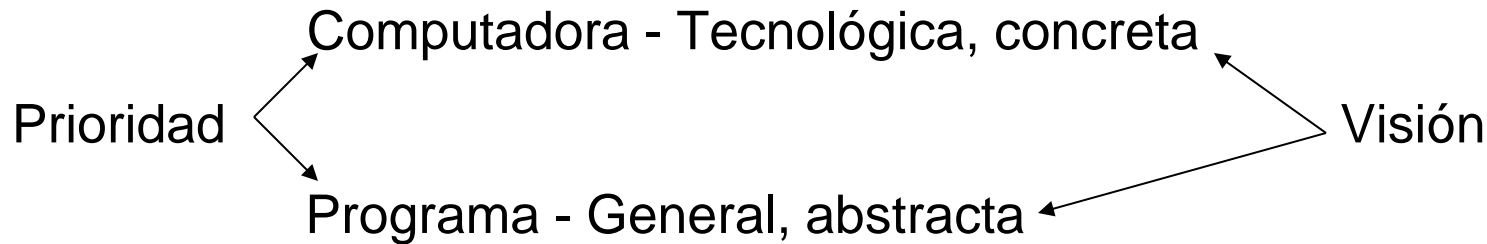


Tema 1:

Introducción

Programas de computadora y Lenguajes de programación

- Programa de Computadora



Compromiso vigencia/practicidad

- Perspectiva abstracta

- La noción de programa

- Difícil de definir

- Intento: Descripción de Acción / Actividad
(que hace la actividad reproducible de manera mecánica)

Programas de computadora y Lenguajes de programación (cont.)

- "Subjetividad" de la noción intuitiva de programa
 - Dada una descripción (texto) hay en general quien
 - la entiende como programa, y quien no
 - dado quien la entienda como programa puede ser o no capaz de ejecutarla
 - Ejemplos:
 - Revisar las bujías; si están bien entonces...
 - Ordenar una secuencia de archivos
 - ...
- En todo caso, calificar : "programa para x"
(alguien o algo) capaz de ejecutarlo.

Por ejemplo: ¿quién es x en los cursos prog1 y prog2?

Programas de computadora y Lenguajes de programación (cont.)

- Problema: Acordar una noción (más) objetiva de programa.
Acordar reglas para escribir programas.

Reglas

- notación, sintaxis
- significado (qué acciones son representadas), semántica



Lenguaje de Programación

- Conjunto concreto de programas.
- Decidibilidad: debe ser trivial/mecánico verificar las reglas de notación. ¿Es igual con la semántica?

Programas de computadora y Lenguajes de programación (cont.)


- Modelo de programas
Teoría (Manera de entender el concepto).
- Esto introduce el "problema" del formalismo.
- Los lógicos (matemáticos) en la década del '30 buscaron "modelos universales" de programas.

Autómatas/Máquinas con Estados

- En cada momento, la máquina está en un cierto estado
El conjunto de estados se especifica precisamente
- Hay un conjunto de acciones (transiciones) que son las formas de operar la máquina. Las transiciones cambian el estado de la máquina
- Efecto del programa - consiste en el cambio de estado
Computación por cambio de estado
 - Algunos de los estados posibles contienen información importante; por ejemplo: los estados inicial y final.
- Ver ejemplos!


Ejemplo: SWAP

{ x = X0, y = Y0, temp = ? } Estado inicial

temp = x; 

Ejemplo: SWAP

{ x = X0, y = Y0, temp = ? } Estado inicial

temp = x; 

{ x = X0, y = Y0, temp = X0 }

Ejemplo: SWAP

{ x = X0, y = Y0, temp = ? } Estado inicial

temp = x; ↓


{ x = X0, y = Y0, temp = X0 }

x = y; ↓


{ x = Y0, y = Y0, temp = X0 }

Ejemplo: SWAP

{ x = X0, y = Y0, temp = ? } Estado inicial

temp = x; 

{ x = X0, y = Y0, temp = X0 }

x = y; 

{ x = Y0, y = Y0, temp = X0 }

y = temp; 

{ x = Y0, y = X0, temp = X0 } Estado final

Máquinas de Estado programables

- Los ejemplos vistos son programas individuales
¿Cómo sería una máquina programable capaz de ejecutar CUALQUIER PROGRAMA ?
- Noción universal de Programa
(Programa para la máquina universal)
- Idea:
 - Programa + indicador de hasta dónde se ha ejecutado
 - Esto forma parte del estado (están almacenados en la máquina)

Máquinas de Estado programables (cont.)

- **Esquema:**

- **Componentes:**

- **Memoria**

- Lista de casillas identificadas.
- Cada casilla puede contener un símbolo, de una lista dada.
- Las casillas se usan para almacenar:

Instrucciones y Datos.

- **Punto de control** - "instruction/program counter"

Señala al lugar de memoria donde se encuentra la siguiente instrucción a ejecutar.

- **Instrucciones:**

- hay un conjunto de instrucciones precisamente especificado.
- En general, cada instrucción modifica el estado de la máquina.

Máquinas de Estado programables (cont.)

- **Máquina de Turing**

- 1936 - Modelo de "procedimiento computable"
¿Es decidable mecánicamente si un enunciado matemático es verdadero o falso?

- **Computadora Electrónica**

- Realización del modelo de Turing (~1945)
- Las celdas pueden contener sólo 0 ó 1 (Bits)
- Instrucciones y datos codificados en secuencias de bits (código binario)
- Notar que la noción abstracta de PROGRAMA antecedió a las computadoras electrónicas. Más aún: constituyó la base de su diseño!
- Resultados inesperados de investigaciones altamente teóricas.

Ejemplo: SWAP

				i1					
	x							temp	
		i2		y					
						i3			
PC									

memoria

{ x = X0, y = Y0, temp = ? }

temp = x;

{ x = X0, y = Y0, temp = X0 }

x = y;

{ x = Y0, y = Y0, temp = X0 }

y = temp;

{ x = Y0, y = X0, temp = X0 }

Abstracción

- Queremos programas de computadora para resolver problemas.
- En el enunciado de los problemas se manejan conceptos "abstractos". Por ejemplo: cliente.
- Construir programas para resolver problemas dados involucra representar conceptos abstractos en términos del lenguaje de programación.
- Complejidad de la Programación
 - Inherente a los problemas.
 - Inherente al proceso de construcción de los programas.

Abstracción (cont.)

- Metodologías de Programación. Por ejemplo:

- Partición
- Refinamiento

Uso de Pre y
Poscondiciones



Sinónimo de problema: **ESPECIFICACION**.

El problema especifica qué debe satisfacer el programa buscado, el conjunto de programas aceptables.

- **Partición**

Descomponer la especificación inicial en subproblemas. Verificar que la estructura obtenida resuelve el problema, asumiendo que los componentes están resueltos correctamente.

- **Refinamiento**

Proceder de la misma forma con los componentes aún no implementados

- Analizar como ejemplo el problema de *sorting*.

Abstracción (cont.)

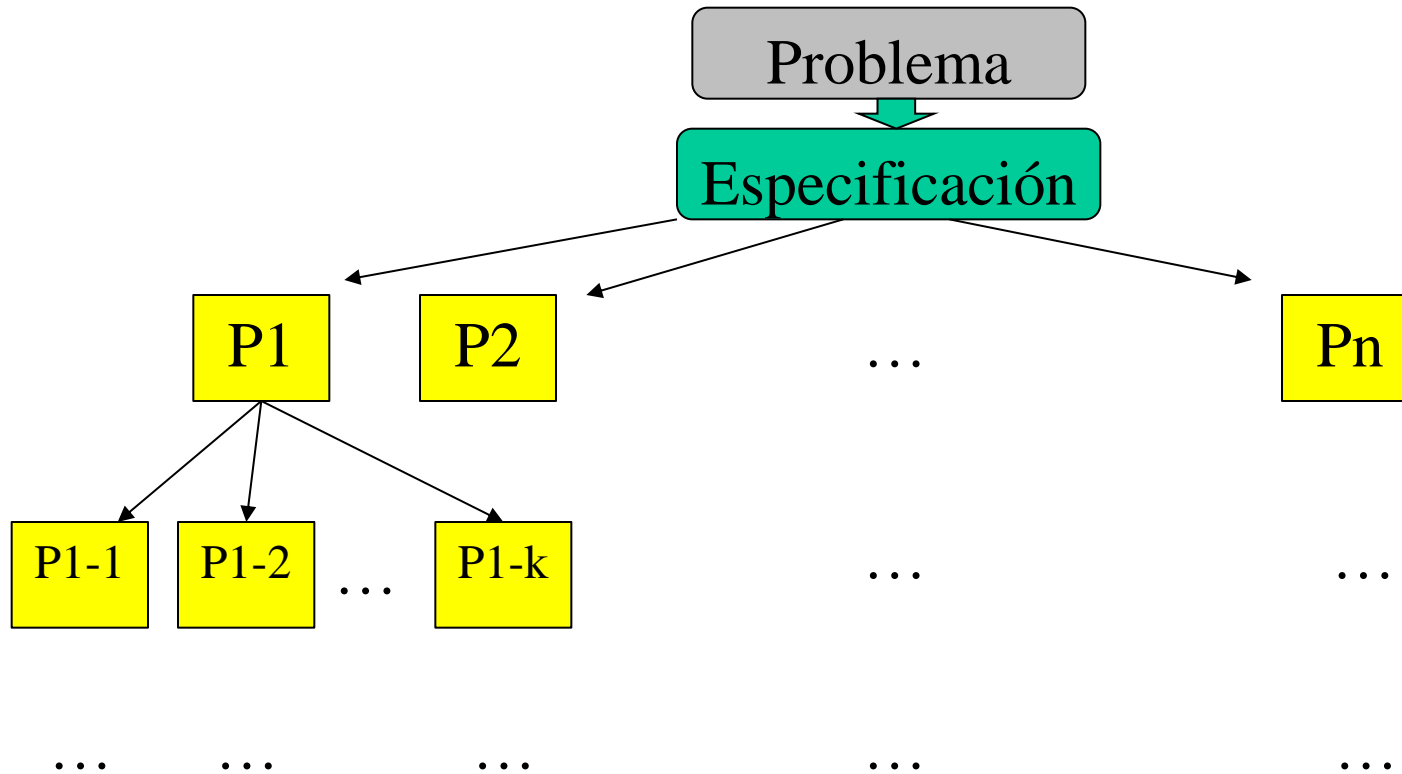
- El proceso sigue refinando conceptos "abstractos" hasta alcanzar el nivel del lenguaje de máquina.
- Ciertas clases de estos refinamientos pueden realizarse automáticamente !!

Entonces es posible definir lenguajes de programación más abstractos.

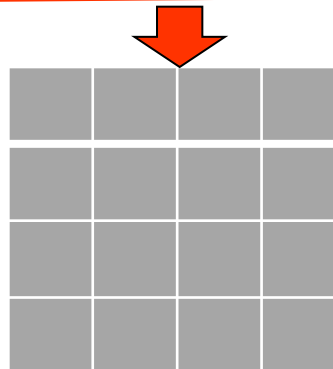
Lenguajes de alto nivel (de abstracción) donde son primitivos ciertos conceptos abstractos.

Hay un programa (compilador / intérprete) que produce representaciones de esos conceptos.

Abstracción: particiones y refinamientos



C++



Memoria

Lenguajes de programación

Ejemplo: programa que recibe un triángulo (lee las coordenadas de los vértices de un triángulo) y calcula su área.

Podemos expresarlo en un lenguaje de "alto nivel" como C/C++ de la siguiente forma:

```
< importaciones >
< declaraciones >
main () // Area Triang.
< declaraciones >
{
    < instrucciones >
} // Area Triang.
```

Ejemplo: Area de Triángulo

- Podemos comenzar introduciendo el tipo de datos que representará el concepto de punto (en coordenadas cartesianas):

- `< declaraciones >`

```
    struct Punto {  
        float x;  
        float y;  
    }
```

– `< . . . >`

- y luego las variables del programa:

- `< declaraciones . . . >`

```
    Punto p1, p2, p3;
```

– `< . . . >`

Ejemplo: Area de Triángulo (cont.)

- Luego refinamos la parte de instrucciones:
 - `< instrucciones >`
 - `< Leer (p1, p2, p3) >;`
 - `< Desplegar (AreaTri (p1, p2, p3)) >;`
 - `–< . . . >`
- Donde estamos haciendo uso de "instrucciones abstractas"
- Abstractas porque no están (todavía) dadas al nivel concreto de detalle del lenguaje de programación. Pero podemos especificar su efecto con precisión.
- Otra forma de decir lo mismo:
 - son instrucciones (se puede especificar su efecto) pero no son instrucciones concretas del lenguaje.

Ejemplo: Area de Triángulo (cont.)

- El problema original (construir un programa) se resuelve por medio de una estructura cuyos componentes son o bien instrucciones concretas o bien otros programas a ser refinados separadamente.
 - **El método permite acotar el nivel de detalle a ser considerado cada vez.**
- Otra forma de decir lo mismo: se usan abstracciones para particionar el problema dado. Luego se refinan las abstracciones siguiendo el mismo método.

Ejemplo: Area de Triángulo (cont.)

- En el ejemplo introdujimos una función AreaTri que hay que refinar:

– < declaraciones ...>

```
float AreaTri (Punto p1, Punto p2, Punto p3)
{ < declaraciones AreaTri ...>
    < Calcular base = distancia (p1, p2) >;
    < Calcular altura = distancia p3 a p1p2 >;
    return base * altura / 2.0;
}; // end AreaTri.
```

Abstracción Procedural

- El uso de instrucciones abstractas se llama:
 - **ABSTRACCION de PROCEDIMIENTO**
 - **SUBPROGRAMAS**
 - **PROCEDIMENTAL**
 - **(Inglés: PROCEDURAL ABSTRACTION)**

y es la forma más elemental de abstracción.

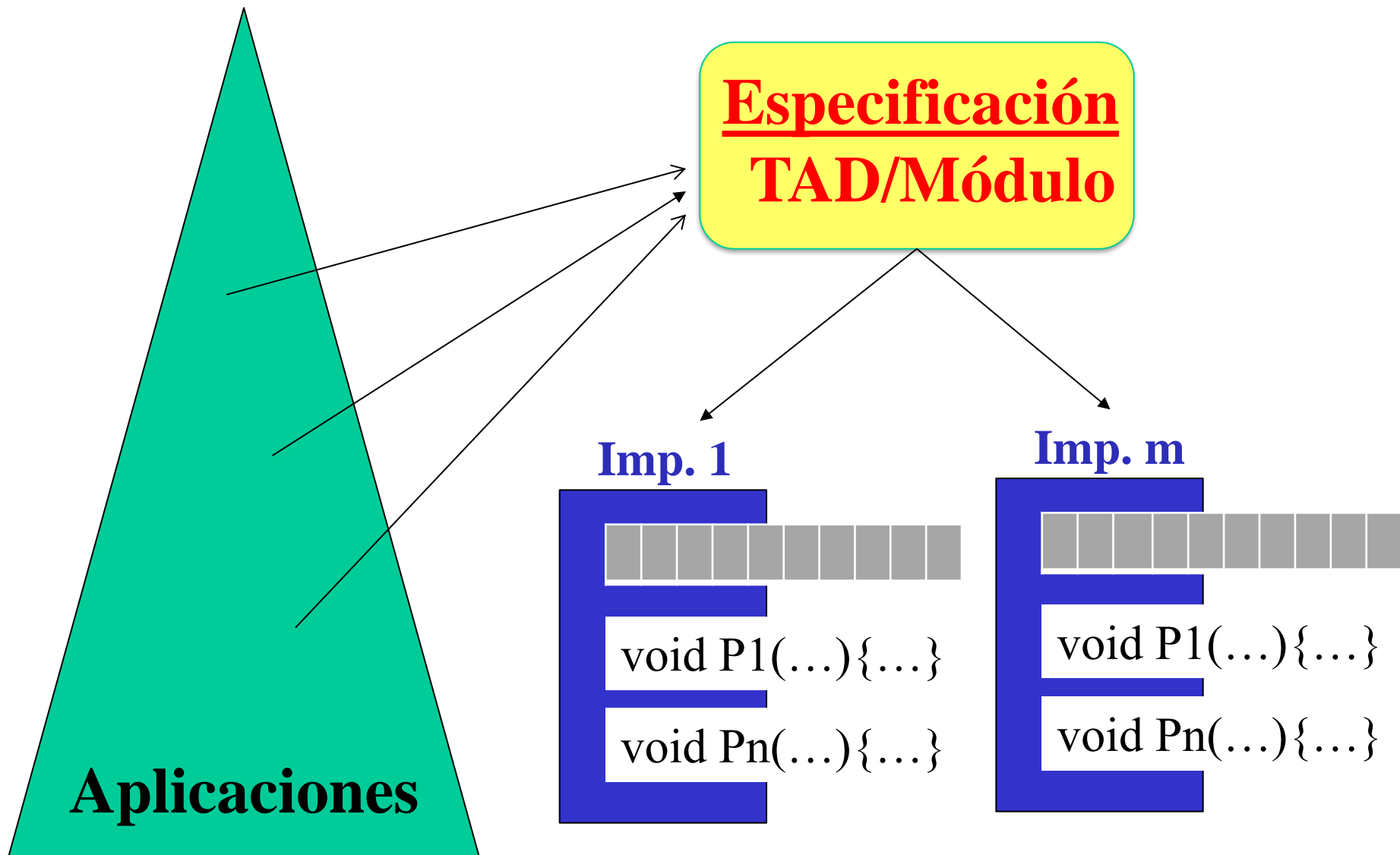
- Algunas instrucciones abstractas se refinan en subprogramas (procedimientos, funciones) en C/C++.
- Otras simplemente en instrucciones que sustituyen textualmente a la instrucción abstracta.

Abstracción de Datos

Más adelante veremos en este curso cómo extender la idea de abstracción a los TIPOS de DATOS.

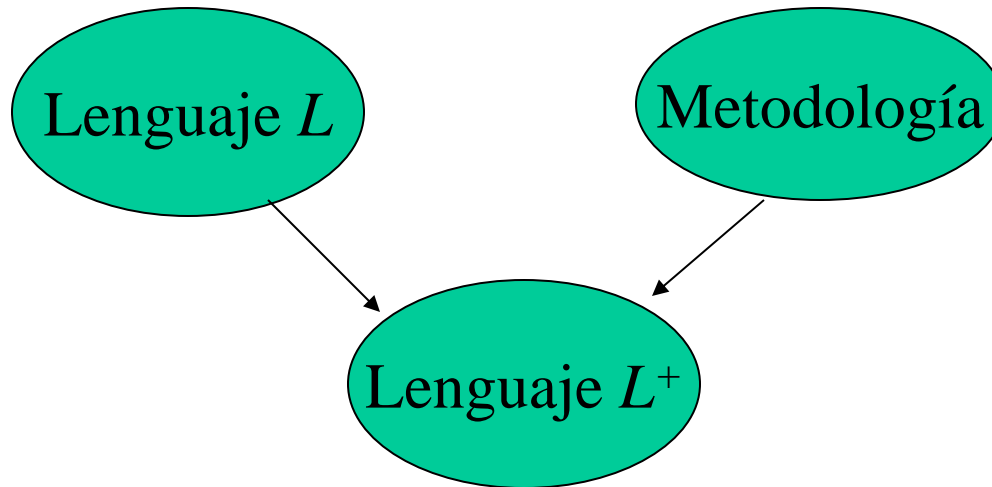
Es decir, usar **TIPOS ABSTRACTOS de DATOS** además de instrucciones abstractas, para diseñar programas (en particular sistemas de porte mediano, tales como un manejador de base de datos o un administrador del file system de un sistema operativo).

Sobre TADs, Abstracción y Modularización



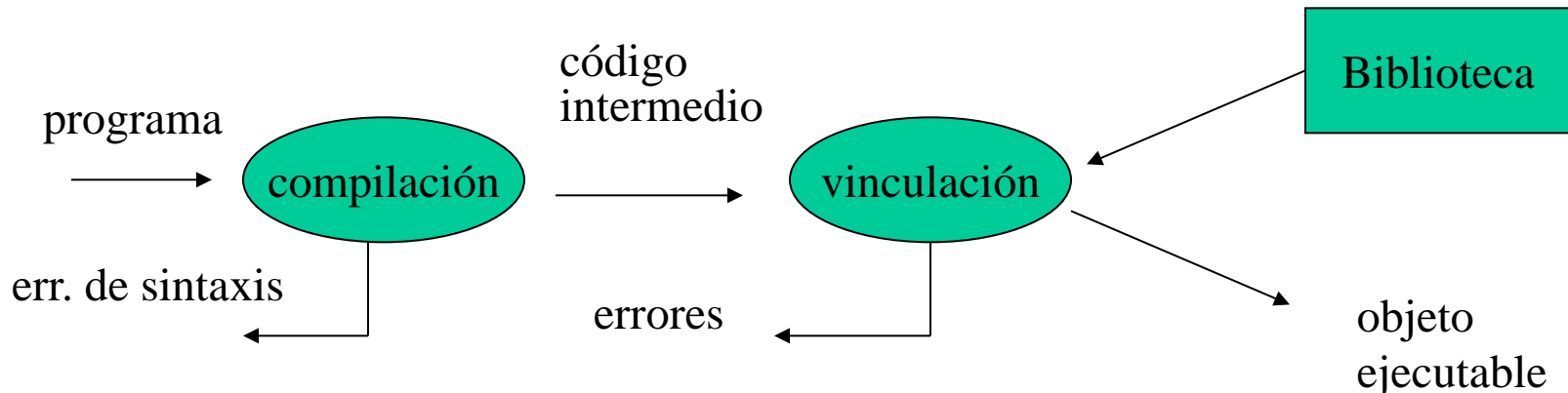
Jerarquía de Lenguajes

- Ciertos conceptos de programación que pueden implementarse en forma automática.
- La jerarquía es potencialmente infinita.
- En la realidad, una especie de caos.



Compilación

- El principio de la compilación separada:
 - A menudo, los programas usan componentes que son contruídos en forma totalmente independiente.
Ej: funciones de biblioteca.
 - Si cambiamos la implementación de una función de biblioteca, no queremos tener que recompilar todos los programas que usan esa función.
 - El programa y (ciertos) componentes auxiliares son compilables separadamente.



Cronograma del curso

