Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-1

Midterm

02 de Octubre 2025

1. Selección múltiple [50 %]

Cada alternativa correcta otorgaba **3 décimas** a esta sección del *midterm*.

1. C
2. C
3. B
4. A
5. E
6. B
7. A o E
8. B
9. C
10. E
11. E
12. C
13. C
14. B
15. C
16. D
17. D
18. A
19. E
20. B

Comentarios

- **Pregunta 7:** Las estructuras de datos que ocasionan el problema descrito en la pregunta son el `set` y `frozenset`, donde ambas estructuras cumplen I, mientras que el `frozenset` además cumple III. Por lo anterior se consideran correctas la alternativas A y E.

2. Desarrollo [50 %]

2.1. Pregunta 1 (Creación de código) [6 puntos]

Consideraciones comunes:

- No se pueden usar estructuras de datos que no fueron nombradas, a excepción de `defaultdict` el cuál si se puede usar.
  - Dado lo anterior, no se pueden usar clases. Si usa clases, no se da puntaje para las estructuras afectadas.
- Para 1A y 1B `dejar_bloque`, se puede entender el parámetro "lugar a dejar" como la base de la torre, o el bloque que está arriba de la torre. Ambos son válidos.
- Para 1A `bloques_ordenados`, se puede entender el "menor a mayor valor" como que la base es el menor valor, o que es el mayor valor. Ambos son válidos.
- Se implementa `stack` como `list`

2.1.1. Parte A [3 puntos]

Desglose puntaje

1. (1 pt.) `tomar_bloque`
   - (0.25) usa `namdetuple` para blocks world con los dos atributos nombrados, y retorna blocks world con los valores actualizados.
   - (0.25) Declara que `bloques` es un `tuple, dict, list` y lo itera apropiadamente.
     - Si es `list o tuple`, la base es el índice de la torre.
   - (0.25) Actualiza la torre sacando el bloque de más arriba siendo la torre un `stack`.
   - (0.25) Declara que `bloques` es un `tuple o dict` y borra la torre de `bloques` cuando se saca la base.
     - Si `bloques` es `tuple`, no se da puntaje si modifica directamente la tupla. Es válido si la tupla tiene listas vacías.
     - Si `bloques` es `dict`, no se descuenta si el código da el error:

       `RuntimeError: dictionary changed size during iteration`
2. (1 pt.) `dejar_bloque`
   - (0.2) usa `namdetuple` para blocks world con los dos atributos nombrados, y lo retorna con los valores actualizados
   - (0.4) Declara que `bloques` es un `tuple, dict, list` y lo usa apropiadamente para encontrar la torre que debe actualizar, o el suelo.
   - (0.4) Deja el bloque sobre el `stack` correcto.

3. (1 pt.) `bloques_ordenados`
   - (0.2) Revisa que la garra esté vacía. `blocks_world` puede ser `namedtuple`, `list`, `dict`
   - (0.4) Declara que `bloques` es un `tuple`, `dict`, `list`, `deque` y revisa que haya una sola torre.
   - (0.4) Revisa que cada torre que tenga bloques esté con los bloques ordenados. Para esta, se permite que torre sea `stack` o `tuple`.

Posible respuesta correcta de 1A con diccionario usando tuplas y el bloque base de la torre como lugar a dejar

```python
BlocksWorld = namedtuple("BlocksWorld", ['garra', 'bloques'])

def tomar_bloque(bloks_world, bloque_escogido):
    _, bloques = blocks_world
    for torre in bloques:
        if torre[-1] == bloque_escogido:
            torre.pop()
    return BlocksWorld(bloque_escogido, bloques)

def dejar_bloque(bloks_world, lugar_a_dejar):
    garra, torres = blocks_world
    if lugar_a_dejar == 0:
        lugar_a_dejar = garra
    torres[lugar_a_dejar].append(garra)
    return BlocksWorld(0, torres)

def bloques_ordenados(blocks_world):
    garra, torres = blocks_world
    if garra != 0:
        return False
    torre = torres[1]
    for base, torre in enumerate(torres):
        if len(torre) > 0 and torre != 1:
            return False
        for i, bloque in enumerate(torre, 1):
            if i != bloque:
                return False
    return True


```

Posible respuesta correcta de 1A con diccionario usando el bloque de arriba de la torre como lugar a dejar

```
BlocksWorld = namedtuple("BlocksWorld", ['garra', 'bloques'])

def tomar_bloque(bloks_world, bloque_escogido):
    _, bloques = blocks_world
    if bloque_escogido in tuple(bloques.keys()):
        del bloques[bloque_escogido]
    else:
        for torre in bloques.values():
            if torre[-1] == bloque_escogido:
                torre.pop()
    return BlocksWorld(bloque_escogido, bloques)

def dejar_bloque(bloks_world, lugar_a_dejar):
    garra, bloques = blocks_world
    if lugar_a_dejar == 0:
        bloques[garra] = [garra]
    else:
        for base_torre, torre in bloques.items():
            if torre[-1] == lugar_a_dejar:
                torres[base_torre].append(garra)
    return BlocksWorld(0, torres)

def bloques_ordenados(blocks_world):
    garra, torres = blocks_world
    if garra != 0:
        return False
    base_es_1 = 1 in torres.keys()
    if len(torres) != 1 or not base_es_1:
        return False
    torre = torres[1]
    for i, bloque in enumerate(len(torre)):
        bloque_esperado = i+2
        if bloque != bloque_esperado:
            return False
    return True

```

2.1.2. Parte B [3 puntos]

Desglose puntaje

1. (1 pt.) `tomar_bloque`
   - (0.4) itera sobre el `frozenset` para encontrar lo que debe sacar.
   - (0.4) Saca el par de valores correcto. Este par es `namedtuple o tuple`
   - (0.2) Deja la garra con el bloque. `blocks_world` debe ser `namedtuple`
2. (1 pt.) `dejar_bloque`
   - (0.8) agrega el par de valores correcto al `frozenset` para encontrar el par a sacar. Este par es `namedtuple o tuple`
   - (0.2) Retorna `blocks_world` con la garra vacía y los otros cambios. `blocks_world` debe ser `namedtuple`
3. (1 pt.) `bloques_ordenados`
   - 1.0 pt, si hace la comparación == de solo los parámetros que recibe la función, y `blocks_world` es *hasheable* dado el formato de estructura que indicó. Casos válidos
     - `blocks_world` es `tuple o namedtuple`
     - `bloques` es `frozenset` que contiene `tuple o namedtuple`
   - 0.5 pt, si solo hace la comparación con `garra` y `bloques` por separado, ya que aprovecha que sea *hasheable* el `frozenset`. Estructuras válidas para este caso:
     - `blocks_world` es `tuple, namedtuple, dict, list, o deque`
     - `bloques` es `frozenset` que contiene `tuple o namedtuple`
   - 0.25 pt, si itera correctamente las estructuras correctas comparando valor a valor. Estructuras válidas para este caso:
     - `blocks_world` es `namedtuple`
     - `bloques` es `frozenset` que contiene `tuple o namedtuple`

Posible Respuesta

Un código Python correcto:

```python
BlocksWorld = namedtuple("BlocksWorld", ['garra', 'bloques'])

def tomar_bloque(bloks_world, bloque_escogido):
    _, bloques = blocks_world
    nuevos_bloques = bloques
    for bloque_arriba, bloque_abajo in bloques:
        if bloque_arriba == bloque_escogido:
            nuevos_bloques = bloques - (bloque_abajo, bloque_arriba)
    return BlocksWorld(bloque_escogido, nuevos_bloques)

def dejar_bloque(bloks_world, lugar_a_dejar):
    garra, bloques = blocks_world
    nuevos_bloques = bloques | (garra, lugar_a_dejar)
    return BlocksWorld(0, nuevos_bloques)

def bloques_ordenados(blocks_world, blocks_world_esperado):
    return blocks_world == orden_esperado

```

2.2. Pregunta 1 (Modelación OOP) [6 puntos]

Clases Cancha [3pts]

Desglose puntaje

- **[0.5 pts]** Definir `Cancha` como abstracta
- **[0.5 pts]** Definir `canchaFutbol`, `CanchaTenis`, `CanchaBasquetbol` como clases que heredan `Cancha`
- **[0.5 pts]** Definir atributos de `cancha`. Debe haber al menos atributos para el responsable (`Persona`), lista de personas, y hora de inicio y termino de arriendo. Precio debería ir aquí, pero también podría estar en las clases derivadas.
- **[0.5 pts]** Definir el método que hace el cobro como abstracto, y definirlo en cada una de las clases derivadas.
- **[0.5 pts]** Clase `CanchaTenis` debe tener una manera de forzar que los arriendos sean por turnos y no por solo por horas.
- **[0.5 pts]** Clase `Cancha` debe contener `Persona`

Clase Persona [2pts]

Desglose puntaje

- **[0.4pts]** Debe haber una clase principal `Persona` o equivalente.
- **[0.4pts]** Debe haber una clase `Socio` que hereda de `Persona`
- **[0.4pts]** Debe haber una clase `DeportistaAR` que herede de `Persona`
- Alternativamente pueden ser 3 clases del mismo nivel (Persona, Socio, DeportistaAR(que heredan de una clase más general
- **[0.4pts]** Las personas deben tener al menos un identificador o nombre, y una manera de obtener su edad.
- **[0.4pts]** En el caso de los DeportistaAR, debe tener una manera identificar qué tipo de deportes practica.

Clase DCC [1pto]

Desglose puntaje

- **[0.6pts]** La clase `DCC` debe contener `Cancha`s
- **[0.2pts]** La clase `DCC` debe contener horario de apertura y cierre
- **[0.2pts]** La clase `DCC` debe contener métodos que permitan obtener los reportes pedidos.

Posible Respuesta