UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 2
Pregrado
2022-II
Profesor: José Fiestas
Lab: 1.03

Indicaciones específicas:

- Esta evaluación contiene 8 páginas (incluyendo esta página) con 3 preguntas. El total de puntos son 20.
- El tiempo límite para la evaluación es 100 minutos.
- Cada pregunta deberá ser respondida en un solo archivo con el número de la pregunta.
  - `p1.cpp`
  - `p2.cpp`
  - `p3.cpp`
- Deberás subir estos archivos directamente a www.gradescope.com, uno en cada ejercicio. También puedes crear un .zip
- **No está permitido** copiar soluciones de fuentes externas. Si se detecta entregas con soluciones comunes, se anulará la prueba.

Competencias:

- Para los alumnos de la carrera de Ciencia de la Computación

  Aplicar conocimientos de computación apropiados para la solución de problemas definidos y sus requerimientos en la disciplina del programa. (nivel 2)

  Diseñar, implementar y evaluar soluciones a problemas complejos de computación.(nivel 2)

  Crear, seleccionar, adaptar y aplicar técnicas, recursos y herramientas modernas para la práctica de la computación y comprende sus limitaciones. (nivel 2)

- Para los alumnos de las carreras de Ingeniería

  Aplicar conocimientos de ingeniería en la solución de problemas complejos de ingeniería (nivel 2).

  Diseñar soluciones relacionados a problemas complejos de ingeniería (nivel 2)

  Crear, seleccionar y utilizar técnicas, habilidades, recursos y herramientas modernas de la ingeniería y las tecnologías de la información, incluyendo la predicción y el modelamiento, con la comprensión de sus limitaciones (nivel 2)

- Para los alumnos de Administración y Negocios Digitales

  Analizar información verbal y/o lógica proveniente de distintas fuentes, encontrando relaciones y presentándola de manera clara y concisa (nivel 2)

  Analizar y evaluar el comportamiento del consumidor y el desarrollo de estrategias comerciales (nivel 2)

  Trabajar de manera efectiva con equipos multidisciplinarios y diversos en género, nacionalidad, edad, etc. (nivel 2)

Calificación:

Tabla de puntos (sólo para uso del professor)

Question	Points	Score
1	7	
2	6	
3	7	
Total:	20	

1. (7 points) **Matriz booleana**

   Dado un array de tamaño n, que llamaremos **primer array**, escriba un código que construya una **matriz dinámica** de tamaño n×n, ejecutando los pasos que se indican

   - cada celda de la matriz tiene valor 1 si el valor del primer array en la **posicion i** es mayor o igual que el valor del mismo array en la **posicion j**. *Por ejemplo, si el primer array es {2,5,1,3,6,8,7,9,4,1}, la matriz en la posición (3,2) tiene valor 1, la matriz en la posición (3,4) tiene valor 0*
   - imprima la matriz obtenida
   - genere un **segundo array** cuyos elementos son la suma de los valores de cada fila de la matriz del paso anterior. *Por ejemplo, si la fila 2 (empezando de cero) de la matriz, tiene valores {1,0,1,1,0,0,0,0,0,1}, el valor del* **segundo array** *en la posicion 2 (array[2]) seria igual a 4*
   - determine la **posición del valor máximo** del **segundo array** e imprima el valor del **primer array** en esa posición. *Por ejemplo, si el mayor valor del segundo array está en la posición 7, se imprimirá el valor del primer array en la posición 7, es decir, 9*
   - finalmente, imprima los valores del **primer array** en las posiciones dadas por los valores del **segundo array** en orden ascendente. *Por ejemplo, si el segundo array es {3,6,2,4,7,9,8,10,5,2} , se imprimirá el valor del primer array en la posicion 2, luego el valor en la posicion 9, luego en la posicion 0, luego en la posicion 3, etc.*

   Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Ejecución	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`3pts`)	El diseño del algoritmo es ordenado y claro. La ejecución es correcta (`2pts`)	El diseño tiene algunas deficiencias pero la ejecución es correcta (`1pts`).	El diseño es deficiente y la ejecución no es correcta (`0 pts`)
Sintaxis	No existen errores sintácticos o de compilación (`2pts`)	Existen algunos errores sintácticos de menor relevancia, que no afectan el resultado (`1.5pts`).	Existen errores sintácticos en la forma de ejecución, que no afectan el resultado (`1pts`).	El código tiene errores de sintáxis que afectan el resultado (`0.5pts`)
Optimizacion	El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pts`)	El código es de buen performance durante la ejecución (`1.5pts`)	El código no está optimizado pero la ejecución no es deficiente(`1pts`)	El código no está optimizado y la ejecución es deficiente (`0pts`)

2. (6 points) **Cadena de farmacias**

Una cadena de farmacias debe repartir 5000 productos con fechas de vencimiento, entre 2022 y 2032. Inicialmente, todos los productos están en almacén.
Se requiere distribuir los productos en locales, con capacidad máxima de 2000 productos cada uno, de la siguiente manera:

- enviar a un local aquellos con fecha de vencimiento entre 2022 y 2025
- enviar a un segundo local aquellos entre 2026 y 2030
- enviar el resto a otro local
- se decide luego separar los productos con fecha de vencimiento par e impar del segundo local (2026-2030) y disponerlos en dos locales separados
- ingresa un nuevo lote de 2000 productos, y se reparte de nuevo en pares en un local e impares en otro

Las fechas de vencimiento pueden ser random con los limites indicados.
Note que si un local excede su capacidad, se debe usar otro local.
El encargado de almacén solicita espacio en 7 locales para la distribución. Se pide determinar, utilizando un código en C++, y representando a los locales con vectores, cuántos locales se usarán para la distribución final de productos y cuántos no tendrán que ser ocupados.

Por ejemplo:
local 1: 1830 productos
local 2: 2000 productos
local 3: 434 productos
local 4: 1814 productos
local 5: 922 productos
Se usarán 5 locales de 7

Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Ejecución	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`3pts`)	El diseño del algoritmo es ordenado y claro. La ejecución es correcta (`2pts`)	El diseño tiene algunas deficiencias pero la ejecución es correcta (`1pts`).	El diseño es deficiente y la ejecución no es correcta (`0 pts`)
Sintaxis	No existen errores sintácticos o de compilación (`2pts`)	Existen algunos errores sintácticos de menor relevancia, que no afectan el resultado (`1.5pts`).	Existen errores sintácticos en la forma de ejecución, que no afectan el resultado (`1pts`).	El código tiene errores de sintáxis que afectan el resultado (`0.5pts`)
Optimizacion	El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pts`)	El código es de buen performance durante la ejecución (`1.5pts`)	El código no está optimizado pero la ejecución no es deficiente(`1pts`)	El código no está optimizado y la ejecución es deficiente (`0pts`)

3. (7 points) **Dodecaedro**

Genere una clase Dodecaedro (dado de 12 caras) e implemente el siguiente juego:

- Por cada ronda, cada jugador señala la cantidad de intentos en las que quiere lanzar el dado (entre 1 y 3)
- cada vez que lanza el dado, debe indicar el numero que debe salir (entre 1 y 12)
- en cada lanzada se contabilizan puntos que equivalen al valor absoluto de la diferencia entre lo esperado y el número que salió
- El numero que sale del dado debe ser random (entre 1 y 12), y el numero esperado en el primer intento (entre 1 y 12) debe ser ingresado por el usuario. Los siguientes intentos tienen números esperados consecutivos al primero (ver ejemplo)
- Finalmente, se suman los puntos de cada jugador en todas las rondas, y ganara el que tiene menor cantidad de puntos

Se debe simular una partida de 3 jugadores y 3 rondas de juego.
La clase debe contar con los atributos que considere necesarios, por lo menos con dos constructores y un destructor
Imprima en cada ronda: **el nombre del jugador**, y por cada intento, el **número que salió**, el **numero esperado** y el **puntaje por lanzamiento**. Finalmente el nombre del ganador.
Por ejemplo, para la primera ronda:
Jugador 1, ¿Cuántas veces tirará el dado?
3
primer intento, 7, 3, 4
segundo intento, 4, 4, 0
tercer intento, 3, 5, 2
Puntaje: 6

Jugador 2, ¿Cuántas veces tirará el dado?
2
primer intento, 1, 10, 9
segundo intento, 4, 11, 7
Puntaje: 16

Jugador 3, ¿Cuántas veces tirará el dado?
2
primer intento, 1, 8, 7

segundo intento, 4, 9, 5
Puntaje: 12

ganador es: jugador 1

Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Ejecución	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`3pts`)	El diseño del algoritmo es ordenado y claro. La ejecución es correcta (`2pts`)	El diseño tiene algunas deficiencias pero la ejecución es correcta (`1pts`).	El diseño es deficiente y la ejecución no es correcta (`0 pts`)
Sintaxis	No existen errores sintácticos o de compilación (`2pts`)	Existen algunos errores sintácticos de menor relevancia, que no afectan el resultado (`1.5pts`).	Existen errores sintácticos en la forma de ejecución, que no afectan el resultado (`1pts`).	El código tiene errores de sintáxis que afectan el resultado (`0.5pts`)
Optimizacion	El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pts`)	El código es de buen performance durante la ejecución (`1.5pts`)	El código no está optimizado pero la ejecución no es deficiente(`1pts`)	El código no está optimizado y la ejecución es deficiente (`0pts`)