Segundo Parcial

2 de abril de 2020

Indicaciones generales

1. Este es un examen **individual** con una duración de **120 minutos: de 7:00 a 9:00 am**.
2. En `e-aulas` puede acceder a las diapositivas y a la sección correspondiente a este parcial.
3. Solamente será posible tener acceso a `e-aulas.urosario.edu.co` y a los sitios web correspondientes a la documentación de C++ dispuestos por el profesor.
4. Maletas, morrales, bolsos, etc. deben estar ubicados al frente del salón.
5. Celulares y otros dispositivos electrónicos deben estar apagados y ser guardados dentro de las maletas antes de ser ubicadas en su respectiva posición.
6. El estudiante no debe intentar ocultar ningún código que no sea propio en la solución a la actividad.
7. El estudiante solo podrá disponer de hojas en blanco como borrador de apuntes (opcional).
8. El estudiante puede tener una hoja manuscrita de resumen (opcional). Esta hoja debe estar marcada con nombre completo.
9. `e-aulas` se cerrará a la hora en punto acordada. La solución de la actividad debe ser subida antes de esta hora. El material entregado a través de `e-aulas` será calificado tal como está. Si ningún tipo de material es entregado por este medio, la nota de la evaluación será 0.0. **Se aconseja subir a e-aulas versiones parciales de la solución a la actividad.**
10. Todas las evaluaciones serán realizadas en el sistema operativo GNU/Linux.
11. Todas las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
12. La evaluación debe presentarse exclusivamente en uno de los computadores ubicados en el salón de clase y a la hora acordada. Presentar la evaluación desde otro dispositivo o en otro horario diferente al estipulado es causa de anulación.
13. **Cualquier incumplimiento de lo anterior conlleva la anulación del examen.**
14. Las respuestas deben estar totalmente justificadas.
15. **Entrega**: archivos con extensión `.txt`, `.cpp` o `.hpp` según el caso, conteniendo la demostración o el código. Nombre los archivos como `pY.Z`, con `Y = 1,2,3` y `Z = txt, cpp, hpp`. Comprima su código y demás archivos en *un único* archivo `parcial.zip` y súbalo a `e-aulas`. **Importante:** no use acentos ni deje espacios en los nombres de los archivos que cree.

En el desarrollo del examen, no olvide usar la plantilla para cada ejercicio.
Las implementaciones deben ejecutarse sin errores usando las funciones `main()` de cada plantilla.

1. [20 ptos.] [*Collatz conjecture*.] Conocida también como conjetura $3n+1$ o conjetura de Ulam, fue propuesta por el matemático Lothar Collatz en 1937, y a la fecha no se ha resuelto. La conjetura se define así: Suponga que la siguiente operación se aplica sucesivamente a cualquier número entero positivo n:

   - Si el número n es par, se divide entre 2
   - Si el número n es impar, se multiplica por 3 y se suma 1

   La conjetura es: *No importa el valor inicial de n, la sucesión siempre llegará a* 1. Por ejemplo, empezando con $n = 6$, se obtiene la sucesión 6, 3, 10, 5, 16, 8, 4, 2, 1. Implemente una versión recursiva de la conjetura de Collatz como la función `void collatz(int n)` que imprime la sucesión resultante, y verifique su funcionamiento.

2. [40 ptos.] Los profesores de matemáticas del departamento MACC están desarrollando un software para las clases de geometría en la Universidad. Como monitor es su tarea implementar la parte del software que tiene que ver con polígonos regulares. Recuerde que un polígono regular es un polígono cuyos lados y ángulos interiores son iguales entre sí. Para desarrollar el módulo de polígonos regulares, usted debe escribir código que implementa esta idea geométrica.

   Como un primer acercamiento al software final, implemente una clase `RegularPolygon` que tiene las características descritas a continuación. Los atributos deben ser

   - `n_sides`: que representa el número de lados de polígono regular
   - `side_len`: que representa la longitud de los lados del polígono regular

   por otro lado, los correspondientes métodos de la clase son

   - `RegularPolygon`: constructor que recibe el número de lados y la longitud de un lado
   - `get_xxx`: métodos *getters* para cada uno de los atributos de la clase
   - `perimeter`: que calcula y retorna el perímetro del polígono regular
   - `area`: que calcula y retorna el área del polígono regular

   Finalmente, sobrecargue los siguientes operadores

   - `operator+`: que recibe dos instancias de la clase `RegularPolygon` llamadas `poly_1` y `poly_2` y retorna una tercera instancia `poly_3`. El objeto `poly_3` debe tener un número de lados igual a la suma de los lados de `poly_1` y `poly_2` y longitud de lado iguale al promedio de la longitud de `poly_1` y `poly_2`. Es decir, si `poly_1` tiene una longitud de lado l_1 y un número de lados n_1; y algo similar para `poly_2` y `poly_3`. Los valores de los atributos de `poly_3` serán

   $$l_3 = \frac{l_1 + l_2}{2}, \qquad n_3 = n_1 + n_2.$$

- `operator*`: que recibe un entero i y una instancia de `RegularPolygon` con n lados y retorna una nueva instancia de `RegularPolygon` con $n * i$ lados y la misma longitud de lado que tenía la instancia que es parámetro. Note que para que este producto sea conmutativo, hay que definir dos versiones del operador: `operator*(RegularPolygon& p , int i)` y `operator*(int i, RegularPolygon& p)`, en donde las dos realizan la misma operación.

Estos operadores **no** deben ser miembros de la clase `RegularPolygon`.

Asegúrese de probar la implementación de su clase antes de que sea integrada al software en desarrollo en el departamento.

3. [40 ptos.] La Universidad le encarga optimizar la velocidad de sus enrutadores de modo que el acceso a las clases virtuales sea más rápido. Cada enrutador almacena los nombres de los salones virtuales y la cantidad de clases que tienen lugar en cada salón. La estrategia para aumentar la velocidad de los enrutadores es mediante una organización de los salones virtuales de acuerdo a la cantidad de clases que se dicta en cada uno de ellos. Sin embargo, el numero de salones puede llegar a ser muy grande; por lo tanto, usted decide organizar los salones que tienen una cantidad de clases igual o mayor a una variable `range`, dejando desorganizados los salones virtuales que tienen un número de clases estrictamente menor a dicho rango.

   Suponga que los salones y la cantidad de clases están guardados como un `vector<string>` y un `vector<int>`, respectivamente. Usando una variante de `selection sort`, escriba una función

```
1 void router_speedup(vector<string> &rooms, vector<int> &numcls,
     int range);
```

   que implementa la estrategia de optimización discutida arriba. Por ejemplo, si los vectores de salones y número de clases son

```
rooms = {"Tesla", "Lovelace", "Hipatia", "Caldas", "Neumann", "
   Casur303", "Casur666"}
numcls = {3, 8, 6, 1, 1, 2, 12}
```

   el resultado de la estrategia de optimización es

```
rooms = {"Casur666", "Lovelace", "Hipatia", "Tesla", "Caldas", "
   Neumann", "Casur303"}
numcls = {12, 8, 6, 3, 1, 1, 2}
```

   donde el rango seleccionado fue `range = 3`.