Segundo Parcial

17 de octubre de 2020

Indicaciones generales

1. Este es un examen **individual** con una duración de **100 minutos: de 10:10am a 11:50am**.
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. [40 ptos.] Sea H_n^a la siguiente suma parcial finita

$$H_n^a = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \cdots + \frac{(-1)^{n+1}}{n}.$$

Escriba una función recursiva `float alt_har_num(const int n)` que calcula el n-ésimo número H_n^a y lo retorna como un tipo de dato flotante.

Ayuda: Puede resultar útil implementar primero una función recursiva que calcula primero los términos de la suma parcial finita

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \cdots + \frac{1}{n},$$

donde no hay alternancia de signo ni subíndice a en la definición. Una vez implemente la función recursiva para H_n, unos pocos cambios generarán la solución recursiva para H_n^a.

La correctitud de la solución puede ser parcialmente verificada notando que $H_n^a \to \ln 2$, cuando $n \to \infty$. Es decir, si escoge valores grandes de n su solución debería ser cercana a $\ln 2$.

2. A continuación se presenta la interfase para la clase `Vector3D`, que usando un arreglo dinámico implementa vectores en 3 dimensiones, junto con las operaciones de producto por un escalar y producto punto entre vectores, que se implementan como funciones en lugar de métodos:

```
class Vector3D{
private:
  double *vec;

public:
  Vector3D();
  ~Vector3D();

  Vector3D& operator=(const Vector3D &rhs);
  double operator[](int i) const; //get operator
  double& operator[](int i); //set operator
};

//vector-scalar product
Vector3D operator*(const Vector3D& v, double c);
Vector3D operator*(double c, const Vector3D& v);
//dot product
double operator*(const Vector3D& v1, const Vector3D& v2);
```

Se provee el archivo `vector3d_plantilla.cpp`, al que hay que renombrar `vector3d.cpp` y realizar las siguientes implementaciones:

a) [30 ptos.] Implemente los métodos constructor y destructor de la clase.

b) [30 ptos.] Implemente las sobrecargas del `operator*` para obtener el producto de escalar por vector y su versión simétrica, así como la de producto punto entre vectores.