UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 1
Pregrado
2022-I
Profesor Jorge Villavicencio
Lab 1.01

Indicaciones específicas:

- Esta evaluación contiene 9 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`
- No se permiten tildes en los resultados en consola.
- Recuerda que el Gradescope solo conserva el último envio que se realiza, por lo tanto una vez que tengas las 3 preguntas resueltas, **deberás arrastrar los 3 archivos de manera simultánea y subirlos al Gradescope**.
  www.gradescope.com

Competencias:

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

  Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina. (Evaluar)

  Analizar problemas e identificar y definir los requerimientos computacionales apropiados para su solución.(Usar)

  Utilizar técnicas y herramientas actuales necesarias para la práctica de la computación. (Usar)

- Para los alumnos de las carreras de Ingeniería

  Capacidad de aplicar conocimientos de matemáticas (nivel 3)

  Capacidad de aplicar conocimientos de ingeniería(nivel 2)

  Capacidad para diseñar un sistema, un componente o un proceso para satisfacer las necesidades deseadas dentro de restricciones realistas (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) **Ecuación cuadrática para encontrar raíces solución.**
Dada la ecuación cuadrática:

$$ax^2 + bx + c = 0$$

donde a, b, c son los coeficientes de la ecuación.
Tenemos la fórmula cuadrática para obtener las raices solución:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

donde la discriminante es:

$$D = b^2 - 4ac$$

Según el valor de la discriminante se puede determinar los siguiente casos:

1. D > 0 , se obtiene dos raices solución.
2. D $= 0$, se obtiene una raiz solución.
3. D < 0 , no se puede obtener raices solución.

Se pide crear un programa que permita al usuario ingresar los valores de los coefecientes a, b y c de la ecuación cuadrática e indicar sus raices solución.

Consideraciones:

- El coeficiente "a"no puede tener el valor 0, en caso el usuario ingrese el número 0, el programa debe solicitar nuevamente el valor del coeficiente.
- En caso la D > 0 el programa debe imprimir "Existe 2 raices solucion". Imprimir los valores de las raices solución x1 y x2.
- En caso la D $= 0$ el programa debe imprimir "Existe 1 raiz solucion". Imprimir el valor de la raíz solución.
- En caso la D < 0 el programa debe imprimir "No existe raices solucion".
- Considerar una precisión decimales de 3.
- Las únicas librerías permitidas son iomanip y cmath.

Listing 1: Ejemplo 1

```
a = 1
b = -2
c = 1
Existe 1 raiz solucion
x1 = x2 = 1.000
```

Listing 2: Ejemplo 2

```
a = 1
b = 2
c = 1
```

```
Existe 1 raiz solucion
x1 = x2 = -1.000
```

Listing 3: Ejemplo 3

```
a = 6
b = 10
c = -1
Existe 2 raices solucion
x1 = 0.095
x2 = -1.761
```

Listing 4: Ejemplo 4

```
a = 0
a = 0
a = 0
a = 6
b = 10
c = -1
Existe 2 raices solucion
x1 = 0.095
x2 = -1.761
```

Listing 5: Ejemplo 5

```
a = 5
b = 3
c = 4
No existe raices solucion
```

La rúbrica para esta pregunta es:

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`)	La ejecución es correcta (`1pts`).	La ejecución no es correcta (`0.5pts`)
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 codigo es de buen performance durante la ejecución (`1.5pts`).	El codigo no está optimizado pero la ejecución no es deficiente(`1pts`).	El codigo no está optimizado y la ejecución es deficiente (`0pts`).

2. (6 points) **Fibonacci recursivo.**
Los números fibonacci son una secuencia de números enteros infinitos que se caracterizan porque cada término es la suma de los dos anteriores y se encuentra definido con la siguiente recursividad (caso recursivo):

$$F_n = F_{n-1} + F_{n-2}$$

donde $F_1 = F_2 = 1$ y $F_0 = 0$ (casos base).

$$F_0 = 0$$
$$F_1 = 1$$
$$F_2 = 1$$
$$F_3 = 2$$
$$F_4 = 3$$
$$F_5 = 5$$
$$F_6 = 8$$
$$F_7 = 13$$
$$\ldots$$
$$F_n = F_{n-1} + F_{n-2}$$

Se solicita crear un programa que calcule el número fibonnaci en la posició "n", por ejemplo si n = 7 el resultado es 13. Crear una función recursiva que permita calcular el número Fibonacci.

Consideraciones:

- El rango de valores que puede ingresar para la variable n es entre 0 y 40. El programa debe validar el valor del dato, en caso sea un número fuera del rango se debe volver a solicitar su ingreso

Listing 6: Ejemplo 1

```
n = -42
n = 49
n = 12
F(12) = 144
```

Listing 7: Ejemplo 2

```
n = 0
F(0) = 0
```

Listing 8: Ejemplo 3

```
n = 14
F(14) = 377
```

Listing 9: Ejemplo 4

```
n = 40
F(40) = 102334155
```

La rúbrica para esta pregunta es:

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Ejecución	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`2pts`)	El diseño del algoritmo es ordenado y claro. La ejecución es correcta (`1.5pts`)	La ejecución es correcta (`1pts`).	La ejecución no es correcta (`0.5pts`)
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 codigo es de buen performance durante la ejecución (`1.5pts`).	El codigo no está optimizado pero la ejecución no es deficiente(`1pts`).	El codigo no está optimizado y la ejecución es deficiente (`0pts`).

3. (7 points) **Funciones para un triángulo.**
Realizar un programa que permita ingresar 3 valores con los cuales se debe validar si se pueden formar un triángulo, en caso de poder formar un triangulo se debe calcular el semiperímetro, el área del tríangulo y mostrarlo en consola, en caso de no que no se pueda formar un triángulo con los 3 valores ingresados se debe mostrar el siguiente mensaje en pantalla **"No se puede formar un triangulo"** .

Listing 10: Prototipo de Funciones

```
void leerLado(float &lado);
bool validarExistenciaTriangulo(float a, float b, float c);
void semiPerimetro(float a, float b, float c, float *
   semiPerimetro);
float areaTriangulo(float *a, float *b, float *c,float
   semiPerimetro);
```

Consideraciones:

- Está permitido crear más funciones si el alumno considera que el ejercicio lo requiere, pero como mínimo debe usar las funciones prototipo propuestas.
- La función "validarExistenciaTriangulo", debe validar el Teorema de existencia del triángulo: La suma de las longitudes de cualesquiera dos lados de un triángulo es mayor que la longitud del tercer lado. Dado los lados a, b y c se debe cumplir lo siguiente:

$$a + b > c$$
$$a + c > b$$
$$b + c > a$$

- La función "semiPerimetro", es la sumatoria de los lados entre 2.
- Hacer uso correcto de las funciones y los tipos de parámetros propuestos en los prototipos (valor, referencia y referencia punteros.)
- Considerar una precisión decimales de 3.
- Las únicas librerías permitidas son iomanip y cmath.

Listing 11: Ejemplo 1

```
5
4
3
Semiperimetro = 6.000
Area del triangulo = 6.000
```

Listing 12: Ejemplo 2

```
12
8
```

```
10
Semiperimetro = 15.000
Area del triangulo = 39.686
```

Listing 13: Ejemplo 3

```
5
9
2
No se puede formar un triangulo
```

Listing 14: Ejemplo 4

```
0
0
0
No se puede formar un triangulo
```

La rúbrica para esta pregunta es:

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`)	La ejecución es correcta (`1pts`).	La ejecución no es correcta (`0.5pts`)
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 codigo es de buen performance durante la ejecución (`1.5pts`).	El codigo no está optimizado pero la ejecución no es deficiente(`1pts`).	El codigo no está optimizado y la ejecución es deficiente (`0pts`).