UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 1
Pregrado
2021-II
Profesor: Jorge Villavicencio
Lab 1.07

Indicaciones específicas:

- Esta evaluación contiene 10 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

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) **Información de Índice de Masa Corporal IMC**

El Índice de masa corporal se evalúa con la división del peso en KG y la altura del paciente en metros al cuadrado.

$$IMC = \left(\frac{peso}{altura^2}\right)$$

Se solicita crear un programa que permita indicar en que categoría se encuentra el paciente de acuerdo al rango de IMC.

Figure 1: Tabla de niveles de peso según el IMC

Clasificación	IMC (Kg/m²)	Riesgo
Normal	18.5 - 24.9	Promedio
Sobrepeso	25 - 29.9	Aumentado
Obesidad grado I	30 - 34.9	Moderado
Obesidad grado II	35 - 39.9	Severo
Obesidad grado III	Más de 40	Muy Severo

Adicionalmente en caso el paciente se encuentre en **Sobrepeso ,Obsesidad grado I,II o III** se debe mostrar un mensaje indicando la cantidad de KG que debe bajar para tener el IMC 20.

- El peso se encuentra en gramos.
- La altura inicialmente se encuentra en cm.
- Validar que el peso y la altura sea mayor a 0, en caso se ingrese un número negativo o 0 se debe volver a solicitar su valor.
- Considerar una precisión decimales de 2 para el cantidad de KG que debe bajar el paciente para estar en el estado NORMAL.

Algunos ejemplos en consola de este programa serían:

Listing 1: Ejemplo 1

```
Peso(gr): 68000
Estatura(cm): 165
CLASIFICACION: Sobrepeso - RIESGO: Aumentado
Necesita perder 13.55Kg para IMC 20 NORMAL
```

Listing 2: Ejemplo 2

```
Peso(gr): 50000
```

```
Estatura(cm): 160
CLASIFICACION: Normal - RIESGO: Promedio
```

Listing 3: Ejemplo 3

```
Peso(gr): 80000
Estatura(cm): 168
CLASIFICACION: Sobrepeso - RIESGO: Aumentado
Necesita perder 23.55Kg para IMC 20 NORMAL
```

Listing 4: Ejemplo 4

```
Peso(gr): 60000
Estatura(cm): 170
CLASIFICACION: Normal - RIESGO: Promedio
```

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) **Sumatoria de números al exponente n de forma recursiva.**
Tenemos las siguientes sumatorias de potencias:

$$\sum_{i=1}^{m} \frac{1}{x_i} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{m}$$

$$\sum_{i=1}^{m} \frac{1}{x_i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + ... + \frac{1}{m^2}$$

$$\sum_{i=1}^{m} \frac{1}{x_i^3} = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + ... + \frac{1}{m^3}$$

$$...$$

$$\sum_{i=1}^{m} \frac{1}{x_i^n} = \frac{1}{1^n} + \frac{1}{2^n} + \frac{1}{3^n} + \frac{1}{4^n} + ... + \frac{1}{m^n}$$

Se solicita crear un programa que calcule la sumatoria de la potencia n para el número m. Por ejemplo si la potencia n = 3 y el número m = 6 se tendría la siguiente sumatoria.

$$\sum_{i=1}^{m} \frac{1}{x_i^3} = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + \frac{1}{5^3} + \frac{1}{6^3} = 1.19029$$

Consideraciones:

- Crear una función recursiva que permita calcular la sumatoria de potencias inversas.
- El rango de valores que puede ingresar para la variable **m** es entre 1 y 20. El programa debe validar el valor del dato, en caso sea un número fuera del rango se debe volver a solicitar su ingreso
- El rango de valores que puede ingresar para la variable **n** es entre 1 y 5. El programa debe validar el valor del dato, en caso sea un número fuera del rango se debe volver a solicitar su ingreso

Algunos ejemplos en consola de este programa serían:

Listing 5: Ejemplo 1

```
m = 5
n = 2
Sumatoria de potencia a la 2 hasta el numero (5) = 1.46361
```

Listing 6: Ejemplo 2

```
m = 0
m = 21
m = 6
```

```
n = 3
Sumatoria de potencia a la 3 hasta el numero (6) = 1.19029
```

Listing 7: Ejemplo 3

```
m = 7
n = 20
n = 12
n = 3
Sumatoria de potencia a la 3 hasta el numero (7) = 1.19321
```

Listing 8: Ejemplo 4

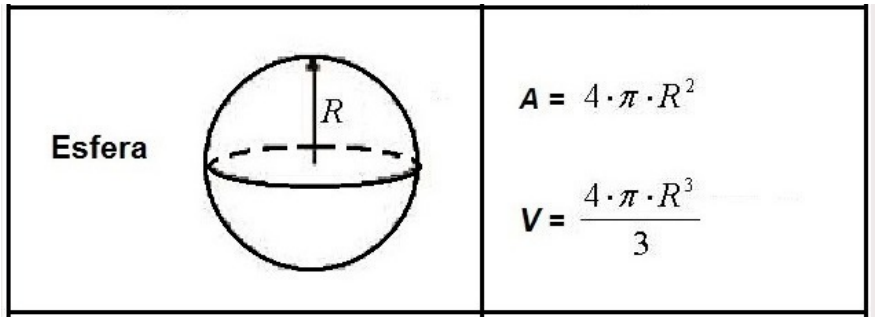
```
m = 20
n = 2
Sumatoria de potencia a la 2 hasta el numero (20) = 1.59616
```

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) **Hallar el área y el volumen de esferas pokemón.**

En un torneo pokemón se está recolectando las características de las pokebolas que tienen forma esférica. Se requiere saber por cada pokebola cual es su área y volumen.



Se solicita crear un programa que permita leer **N** pokebolas. Por cada pokebola se debe solicitar su radio y utilizar las funciones pre establecidas para el área y volumen.

Para la implementación se solicita utilizas como mínimo las siguientes funciones:

Listing 9: Prototipo de Funciones

```
void area_esfera(float &radio, float *area);
void volume_esfera(float *radio, float &volumenEsfera);
```

Consideraciones:

- Definir una variable constante para el valor de PI con 3.1415
- Se debe validar que la cantidad de pokebolas sea mayor a 0, en caso se ingrese un número negativo o 0 se debe volver a solicitar su valor.
- Se debe guardar todas las áreas en un arreglo estático, al terminar el programa se debe mostrar todos las áreas del arreglo en una sola linea como los ejemplos de ejecución.
- Se debe guardar todas los volúmenes en un arreglo estático, al terminar el programa se debe mostrar todos las volúmenes del arreglo en una sola linea como los ejemplos de ejecución.

Algunos ejemplos en consola de este programa serían:

Listing 10: Ejemplo 1

```
Cantidad de pokebolas :5
Valor de PI = 3.1415

Pokebola 1
Radio:2.3

Pokebola 2
Radio:4.3

Pokebola 3
Radio:2

Pokebola 4
Radio:5

Pokebola 5
Radio:12.3

ARREGLO DE AREAS POKEMON
AREAS = [66.472, 232.338, 50.2624, 314.14, 1901.05, ]
ARREGLO DE VOLUMENES POKEMON
VOLUMENES = [50.9619, 333.018, 33.5083, 523.567, 7794.3, ]
```

Listing 11: Ejemplo 2

```
Cantidad de pokebolas :4
Valor de PI = 3.1415

Pokebola 1
Radio:34.2

Pokebola 2
Radio:54.3

Pokebola 3
Radio:3

Pokebola 4
Radio:12

ARREGLO DE AREAS POKEMON
AREAS = [14697.2, 37049.5, 113.09, 1809.45, ]
ARREGLO DE VOLUMENES POKEMON
VOLUMENES = [167548, 670597, 113.09, 7237.79, ]
```

Listing 12: Ejemplo 3

```
Cantidad de pokebolas :0
Cantidad de pokebolas :-4
Cantidad de pokebolas :-64
Cantidad de pokebolas :2
Valor de PI = 3.1414

Pokebola 1
Radio:12.3

Pokebola 2
Radio:23

ARREGLO DE AREAS POKEMON
AREAS = [1901.05, 6647.2, ]
ARREGLO DE VOLUMENES POKEMON
VOLUMENES = [7794.3, 50961.9, ]
```

Listing 13: Ejemplo 4

```
Cantidad de pokebolas :0
Cantidad de pokebolas :5
Valor de PI = 3.1415

Pokebola 1
Radio:12

Pokebola 2
Radio:32.4

Pokebola 3
Radio:34.5

Pokebola 4
Radio:1

Pokebola 5
Radio:0.4

ARREGLO DE AREAS POKEMON
AREAS = [1809.5, 13191.3, 14956.7, 12.566, 2.01056, ]
ARREGLO DE VOLUMENES POKEMON
VOLUMENES = [7238.02, 142466, 172002, 4.18867, 0.268075, ]
```

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`).