UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

# Programación II
## Práctica Calificada 1
## Pregrado
## 2021-II
## Profesor: Jorge Villavicencio
## Lab 2.02

## 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

| ÍNDICE DE MASA CORPORAL (IMC) | CLASIFICACIÓN |
|---|---|
| MENOR A 18.49 | PESO BAJO |
| 18.50 A 24.99 | PESO NORMAL |
| 25 A 29.99 | SOBREPESO |
| 30 A 34.99 | OBESIDAD LEVE |
| 35 A 39.99 | OBESIDAD MEDIA |
| MAYOR A 40 | OBESIDAD MÓRBIDA |

   Adicionalmente en caso el paciente se encuentre en **Obsesidad leve,media o mórbida** 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.

   Algunos ejemplos en consola de este programa serían:

Listing 1: Ejemplo 1

```
Peso ( gr ) :68000
Estatura ( cm ) :165
```

```
CLASIFICACION: Normal
```

Listing 2: Ejemplo 2

```
Peso ( gr ) :89700
Estatura ( cm ) :167
CLASIFICACION: Obesidad leve
Necesita perder 33.922Kg para IMC 20 NORMAL
```

Listing 3: Ejemplo 3

```
Peso ( gr ) :67903
Estatura ( cm ) :155
CLASIFICACION: Sobrepeso
```

Listing 4: Ejemplo 4

```
Peso ( gr ) :120000
Estatura ( cm ) :167
CLASIFICACION: Obesidad morvida
Necesita perder 64.222Kg para IMC 20 NORMAL
```

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 dígitos y factorial recursivo.**

Se solicita crear un programa que calcule la sumatoria de los digitos que conformar al numero **N** por medio de una **función recursiva**. Una vez obtenido la sumatoria de los digitos del número N, se debe determinar el factorial de dicho número utilizando una nueva función recursiva.

**Consideraciones:**

- Crear una función recursiva que permita calcular la sumatoria de los dígitos de un número.
- Crear una función recursiva que permita calcular la el factorial de un número.
- El rango de valores que puede ingresar para la variable **N** es entre 1 y 1000. 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

```
N = 342
Sumatoria de digitos del numero 342 : 9
Factorial (9) : 362880
```

Listing 6: Ejemplo 2

```
N = 34
Sumatoria de digitos del numero 34 : 7
Factorial (7) : 5040
```

Listing 7: Ejemplo 3

```
N = 10001
N = 1001
N = 0
N = -3
N = 345
Sumatoria de digitos del numero 345 : 12
Factorial (12) : 479001600
```

Listing 8: Ejemplo 4

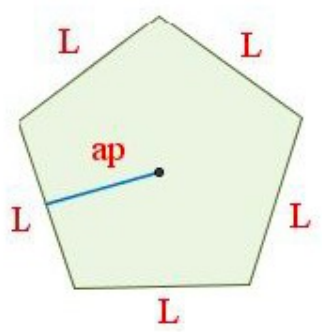
```
N = 234
Sumatoria de digitos del numero 234 : 9
Factorial (9) : 362880
```

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 mayor de un conjunto de polígonos regulares.**

Un polígono regular se caracteriza porque todos sus lados tienen el mismo tamaño. Para poder hallar su perímetro y área tenemos las siguientes fórmulas:




$$P = N * L$$

donde P es el perímetro, N es la cantidad de lados del polígono y L es el tamaño del lado en cm.

$$A = \frac{P * ap}{2}$$

A representa el área del polígono regular y ap es el apotema(distancia del centro del hexágono al punto medio de un lado).

Se solicita crear un programa que permita leer **N** polígonos. Por cada polígono se debe solicitar su tamaño de lado y el tamaño del apotema. Se debe utilizar las funciones pre establecidas para el perímetro y área.

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

Listing 9: Prototipo de Funciones

```
void perimetroPoligono(int n, float *l , float *perimetro);
void areaPoligono(float perimetro, float &apotema, float *
   area);
```

**Consideraciones:**

- Se debe validar que la cantidad de polígonos y el tamaño del lado 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 tal como se muestran en los ejemplos de ejecución.
- Se debe mostrar en consola el área con mayor valor e indicar en que posición se encuentra con respecto al arreglo.

Algunos ejemplos en consola de este programa serían:

Listing 10: Ejemplo 1

```
Cantidad de poligonos regulares :4
Tamanio de Poligono(n) :5

Poligono 1
LADO (l):12
APOTEMA (ap):2

Poligono 2
LADO (l):23
APOTEMA (ap):2

Poligono 3
LADO (l):23
APOTEMA (ap):4

Poligono 4
LADO (l):23
APOTEMA (ap):3

ARREGLO DE AREAS POLIGONO REGULAR  DE 5 LADOS.
AREA [1] = 60
AREA [2] = 115
AREA [3] = 230
AREA [4] = 172.5
```

Listing 11: Ejemplo 2

```
Cantidad de poligonos regulares :0
Cantidad de poligonos regulares :-2
Cantidad de poligonos regulares :2
Tamanio de Poligono(n) :0
Tamanio de Poligono(n) :-4
Tamanio de Poligono(n) :7

Poligono 1
LADO (l):10.2
APOTEMA (ap):3.4

Poligono 2
LADO (l):23.4
APOTEMA (ap):12.3

ARREGLO DE AREAS POLIGONO REGULAR  DE 7 LADOS.
```

```
AREA [1] = 121.38
AREA [2] = 1007.37
El mayor area es 1007.37  y esta en la posicion 2
```

Listing 12: Ejemplo 3

```
Cantidad de poligonos regulares :4
Tamanio de Poligono(n) :9

Poligono 1
LADO (l):12
APOTEMA (ap):3

Poligono 2
LADO (l):12.3
APOTEMA (ap):0.3

Poligono 3
LADO (l):33
APOTEMA (ap):4

Poligono 4
LADO (l):23.96
APOTEMA (ap):3

ARREGLO DE AREAS POLIGONO REGULAR  DE 9 LADOS.
AREA [1] = 162
AREA [2] = 16.605
AREA [3] = 594
AREA [4] = 323.46
El mayor area es 594 y esta en la posicion 3
```

Listing 13: Ejemplo 4

```
Cantidad de poligonos regulares :1
Tamanio de Poligono(n) :10

Poligono 1
LADO (l):123.4
APOTEMA (ap):23.2

ARREGLO DE AREAS POLIGONO REGULAR  DE 10 LADOS.
AREA [1] = 14314.4
El mayor area es 14314.4 y esta en la posicion 1
```

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`). |