UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 1
Pregrado
2022-II
Profesor: Angel Napa
Lab 2.07

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) **Estructuras de control**
   ¿Intervalos disjuntos?
   Formalmente hablando, un intervalo $I = [a, b]$ es el conjunto de todos los números reales x de modo que $a \leq x \leq b$.

$$I = [a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$$

Observe que para que la notación de I sea correcta, es necesario que a sea menor o igual a b. Esto quiere decir que $[3, 1]$ no es un intervalo válido, mientras que $[3, 6]$ y $[4, 4]$ sí son intervalos válidos.

Como I es formalmemte un conjunto, podemos hablar de intersecciones de intervalos como hemos hablado de intersecciones de conjuntos: si I_1 e I_2 son intervalos, $I_1 \cap I_2$ es el conjunto de todos los números reales x de modo que x sea elemento de I_1 e I_2. Formalmente:

$$[a, b] \cap [c, d] = \{x \in \mathbb{R} : a \leq x \leq b \text{ y } c \leq x \leq d\}$$

$$= \{x \in \mathbb{R} : max(a, c) \leq x \leq min(b, d)\} = [max(a, c), min(b, d)]$$

Como ejemplo, tenemos que $[1, 5] \cap [2, 7] = [2, 5]$. Observe que en general la intersección de 2 intervalos puede ser vacía, como es el caso de los intervalos $[1, 2]$ y $[3, 4]$. No es difícil observar que tenemos dos situaciones diferentes: que la intersección de 2 intervalos sea vacía, o que sea otro intervalo.

Se pide construir un programa en C++ que verifique si 2 intervalos cerrados tienen intersección no vacía. Para este problema todos los extremos usados serán enteros. El programa solicitará la lectura de los extremos los intervalos hasta obtener 2 intervalos válidos. Una vez obtenido 2 intervalos válidos, se imprimirá:

- Los dos intervalos recibidos.
- Que la intersección es vacía, si en efecto, los 2 intervalos no comparten algún elemento en común.
- El intervalo que se obtiene al intersecar los 2 intervalos, junto con un elemento cualquiera que pertenezca a esta intersección.

```
Ingrese extremos de I1: 1 4
Ingrese extremos de I2: 2 5

Intervalo 1: [1,4]
Intervalo 2: [2,5]
La interseccion es el intervalo: [2,4]
El punto medio de la interseccion es: 3
```

```
Ingrese extremos de I1: 5 10
Ingrese extremos de I2: 2 8

Intervalo 1: [5,10]
Intervalo 2: [2,8]
La interseccion es el intervalo: [5,8]
El punto medio de la interseccion es: 6.5
```

```
Ingrese extremos de I1: 1 0
Extremos invalidos.
Ingrese extremos de I1: 3 0
Extremos invalidos.
Ingrese extremos de I1: 4 7
Ingrese extremos de I2: 2 3

Intervalo 1: [4,7]
Intervalo 2: [2,3]
Los intervalos tienen interseccion vacia
```

```
Ingrese extremos de I1: 5 4
Extremos invalidos.
Ingrese extremos de I1: 4 10
Ingrese extremos de I2: 2 0
Extremos invalidos.
Ingrese extremos de I2: 4 0
Extremos invalidos.
Ingrese extremos de I2: 5 8

Intervalo 1: [4,10]
Intervalo 2: [5,8]
La interseccion es el intervalo: [5,8]
El punto medio de la interseccion es: 6.5
```

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) **Funciones**
Algoritmo raro
Considere un algoritmo que reciba como dato de entrada un número entero positivo n. Si n es par, el algoritmo lo divide entre dos, y si n es impar, el algoritmo lo multiplica por tres y le agrega 1. El algoritmo se repetirá hasta que eventualmente n tome el valor de 1. Por ejemplo, la secuencia para $n = 3$ será la siguiente:

$$3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Se le pide construir una función *secuencia* que reciba como mínimo un parámetro : el valor de n, e imprima la lista de la secuencia de todos los valores que toma n durante la ejecución del algoritmo mencionado.

```
ingrese n: 1
secuencia:
1
```

```
ingrese n: 3
secuencia:
3 10 5 16 8 4 2 1
```

```
ingrese n: 9
secuencia:
9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

```
ingrese n: 24
secuencia:
24 12 6 3 10 5 16 8 4 2 1
```

```
ingrese n: 27
secuencia:
27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137
412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445
1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438
719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154
3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184
92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
```

```
ingrese n: 2022
secuencia:
2022 1011 3034 1517 4552 2276 1138 569 1708 854 427 1282 641 1924 962 481 1444
722 361 1084 542 271 814 407 1222 611 1834 917 2752 1376 688 344 172 86 43 130
65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 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 (`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) **Punteros**
Casi Máximo.
Escribir dos funciones que realicen lo siguiente:

   1. La primera, *calcularPromedio* recibirá un array (se debe usar punteros) y calculará el promedio de los valores guardados en la lista.
   2. La segunda, *segundoPuesto* recibirá un array (se debe usar punteros) y calculará el segundo máximo elemento del array, si es que este existe(esto es, si es que el array tiene más de un elemento). Si el array tiene solo un elemento, imprimirá un mensaje aclarando que no hay segundo máximo elemento.

   Escriba un programa main que realice lo siguiente:

   - Solicite un valor entero n al usuario. El programa validará que este número deba ser entero positivo.
   - Solicite n enteros positivos que serán guardados en un arreglo de tamaño n.
   - Llame a la función *calcularPromedio* que recibirá el array construido para mostrar el promedio de sus elementos
   - Llame a la función *segundoPuesto* que recibirá el array construido para mostrar en una nueva linea el segundo mayor elemento del arreglo, si es que este existe, caso contrario mostrar un mensaje aclarando que no hay un segundo mayor elemento.

   Aclaración: son libres de ingresar más de un parámetro en las funciones mencionadas. Esto depende de si quisieran trabajar con variables globales, referencia, punteros, etc. Lo que sí es obligatorio es que ambas funciones llamen a un array.

```
Ingresa valor de n: 5
Ingrese valores del array: 1 2 3 4 5

promedio de la lista: 3
puntaje segundo puesto: 4
```

```
Ingresa valor de n: 10
Ingrese valores del array: 3 4 4 2 6 1 7 3 2 10

promedio de la lista: 4.2
puntaje segundo puesto: 7
```

```
Ingresa valor de n: 7
Ingrese valores del array: 4 3 4 3 4 3 4

promedio de la lista: 3.57143
puntaje segundo puesto: 4
```

```
Ingresa valor de n: -1
Valor invalido.
Ingresa valor de n: -4
Valor invalido.
Ingresa valor de n: 0
Valor invalido.
Ingresa valor de n: 5
Ingrese valores del array: 1 1 2 1 1

promedio de la lista: 1.2
puntaje segundo puesto: 1
```

```
Ingresa valor de n: -5
Valor invalido.
Ingresa valor de n: 1
Ingrese valores del array: 1000

promedio de la lista: 1000
No hay segundo maximo
```

```
Ingresa valor de n: 3
Ingrese valores del array: 10 10 10

promedio de la lista: 10
puntaje segundo puesto: 10
```

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`).