UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 2
Pregrado
2021-II
Profesor: Rubén Rivas
Lab: 102

Indicaciones específicas:

- Esta evaluación contiene 8 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`, `p1.h`
  - `p2.cpp`, `p2.h`
  - `p3.cpp`, `p2.h`
- 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 apropiados para la solución de problemas definidos y sus requerimientos en la disciplina del programa. (nivel 2)

  Diseñar, implementar y evaluar soluciones a problemas complejos de computación.(nivel 2)

  Crear, seleccionar, adaptar y aplicar técnicas, recursos y herramientas modernas para la práctica de la computación y comprende sus limitaciones. (nivel 2)

- Para los alumnos de las carreras de Ingeniería

  Aplicar conocimientos de ingeniería en la solución de problemas complejos de ingeniería (nivel 2).

  Diseñar soluciones relacionados a problemas complejos de ingeniería (nivel 2)

  Crear, seleccionar y utilizar técnicas, habilidades, recursos y herramientas modernas de la ingeniería y las tecnologías de la información, incluyendo la predicción y el modelamiento, con la comprensión de sus limitaciones (nivel 2)

- Para los alumnos de Administración y Negocios Digitales

  Analizar información verbal y/o lógica proveniente de distintas fuentes, encontrando relaciones y presentándola de manera clara y concisa (nivel 2)

Analizar y evaluar el comportamiento del consumidor y el desarrollo de estrategias comerciales (nivel 2)

Trabajar de manera efectiva con equipos multidisciplinarios y diversos en género, nacionalidad, edad, etc. (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) **Suma de filas y columnas**

Escribir un programa que utilizando matrices y arreglos dinámicos, lea desde el teclado una matriz de números enteros de tamaño $n \times n$, que calcule la suma de cada fila, almacenado los resultados en un arreglo de tamaño n, de igual modo realizar el mismo cálculo con las columnas, almacenado también los resultados en un arreglo de tamaño n.

Finalmente, el programa debe imprimir la multiplicación de los valores en las posiciones correspondientes de los arreglos generados.

Ejemplo #1:
Input Format:

```
3
2 2 2
3 3 3
4 4 4
```

Constraints:

```
No se requiere textos al ingresar valores (std::cout)
```

Output Format:

```
54 81 108
```

Explicación: Primero se genero los 2 arreglos de tamaño 3 con las sumas correspondiente: el primer arreglo con la suma de filas $6, 9, 12$, y luego el arreglo de la suma de las columnas $9, 9, 9$. Finalmente se multiplicaron los valores de las filas por los valores de las columnas: $6 \times 9 = 54, 9 \times 9 = 81, 12 \times 9 = 108$.

Ejemplo # 2:
Input Format:

```
2
1 1
4 3
```

Output Format:

```
10 28
```

Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado

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`)	El diseño tiene algunas deficiencias pero la ejecución es correcta (`1pts`).	El diseño es deficiente y 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 código es de buen performance durante la ejecución (`1.5pts`)	El código no está optimizado pero la ejecución no es deficiente(`1pts`)	El código no está optimizado y la ejecución es deficiente (`0pts`)

2. (6 points) **Multiplicación del resto**
Escribir un programa que utilizando vectores, lea desde el teclado un vector de números enteros de tamaño n, que reemplace cada valor con el producto del resto de los otros números y muestre los nuevos valores del vector.

Ejemplo #1:
Input Format:

```
6
2 3 1 2 3 4
```

Constraints:

```
No se requiere textos al ingresar valores (std::cout)
```

Output Format:

```
72 48 144 72 48 36
```

Explicación: se tomo el arreglo original y para el primer valor se calculo la multiplicacion del resto que es: $3 \times 1 \times 2 \times 3 \times 4$, lo mismo para el segundo valor: $2 \times 1 \times 2 \times 3 \times 4$, el tercero: $2 \times 3 \times 2 \times 3 \times 4$, asi sucesivamente se obtiene el vector: $\{72, 48, 144, 72, 48, 36\}$

Ejemplo # 2:
Input Format:

```
4
1 1 2 2
```

Output Format:

```
4 4 2 2
```

Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado

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`)	El diseño tiene algunas deficiencias pero la ejecución es correcta (`1pts`).	El diseño es deficiente y 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 código es de buen performance durante la ejecución (`1.5pts`)	El código no está optimizado pero la ejecución no es deficiente(`1pts`)	El código no está optimizado y la ejecución es deficiente (`0pts`)

3. (7 points) **Factorización de polinomio**
Un monomio es una expresión que sigue la siguiente forma: ax^b donde: a es el coeficiente y b el grado.
Definir una clase que represente un monomio (`class monomio_t`) y escribir un programa que permita generar un vector de monomios de tamaño n que lea el coeficiente y el grado de cada uno de los n monomio desde el teclado. El programa deberá generar un nuevo vector que almacene la factorización de los monomios de modo que solo se almacene un monomio por grado. El resultado debera ser mostrado de modo que los monomios se muestren ordenados de menor grado a mayor grado.

Ejemplo #1:
Input Format:

```
5
2 3
1 2
4 3
7 2
4 5
```

Constraints:

```
No se requiere textos al ingresar valores (std::cout)
```

Output Format:

```
8x^2 6x^3 4x^5
```

Explicación: Utilizando el vector original de monomios se genera un nuevo vector que para este ejemplo tendra 3 monomios debido a que solo existen 2 coeficientes (2, 3, 5) y cuyos coeficientes serian (6=2+3, 8=1+7, 5) los cuales se muestran como un polinomio donde los monomios se muestran ordenados por el grado (2, 3, 5).
Ejemplo # 2:
Input Format:

```
6
10 5
1 1
2 2
3 4
2 1
8 5
```

Output Format:

```
3x^1 2x^2 3x^4 18x^5
```

Los criterios en la rúbrica (y el puntaje respectivo) se condicionan a que la solución presentada corresponda al problema planteado

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`)	El diseño tiene algunas deficiencias pero la ejecución es correcta (`1pts`).	El diseño es deficiente y 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 código es de buen performance durante la ejecución (`1.5pts`)	El código no está optimizado pero la ejecución no es deficiente(`1pts`)	El código no está optimizado y la ejecución es deficiente (`0pts`)