UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 2
Pregrado
2022-I
Profesor: Richard Yantas
Lab: 202

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.h`
  - `p2.h`
  - `p3.h`
- Deberás subir estos archivos directamente a canvas cada una de las preguntas `p1.h`, `p2.h` y `p3.h` en formato `pc2.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) **Matrices dinamicas**

   Se pide completar cada una de las funciones mencionadas a continuacion haciendo uso del doble puntero para implementar una matriz.

Funciones
- `void generateMatrix(int **p, int m, int n)`
- `void printMatrix(int **p, int m, int n)`
- `void releaseMemoryMatrix(int **p, int m, int n)`

   - En la funcion `generateMatrix`: Generar una matrix dinamica haciendo uso de **new**[], dado un n, m dimensiones de la matriz y con valores aleatorios entre -10 y 10.
   - En la funcion `printMatrix`: mostrar la matriz separados por un espacio cada elemento en la fila.
   - En la funcion `releaseMemoryMatrix`: hacer uso del `delete[]` operator para liberar memoria.

   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`)
Existen algunos errores sintácticos de menor relevancia, que no afectan el resultado (`1.5pts`).	Sintaxis	No existen errores sintácticos o de compilación (`2pts`)	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) **Vectores**
Aquí el enunciado.
En esta parte se evaluara los conocimientos adquiridos relacionado al manejor de vectores, es importante hace uso de vectores para que pueda ser considerado su solución.

Funciones
- `vector<int> task1(const vector<int> &v)`
- `vector<int> task2(const vector<int> &v)`
- `vector<int> task3(const vector<int> &v)`

- Se pide calcular dado un vector de enteros de entrada obtener un nuevo vector agregando en la secuencua la cantidad de veces que se repite un numero de forma secuencial(es obligatorio usar insert)

  Task 1:

  Input

1 1 1 2 2 3 3 3 3

  Output

1 1 1 3 2 2 2 3 3 3 3 4

- Se pide borrar los elementos negativos impares del vector(es obligatorio usar erase)

  Task 2:

  Input

1 3 4 -2 5 -3 -3 3

  Output

1 3 4 -2 5 3

- Calcula el elemento mayor impar del vector y redimensionar el vector con este ultimo resultado luego si el nuevo vector tiene ceros reemplazarlo por -1 (es obligatorio usar resize)

  Task 3:

  Input

8 3 7 -2 2

  Output

8 3 7 -2 2 -1 -1

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) **Clase CMatrix**

Se pide crear una clase **CMatrix** con un atributo doble puntero ****p**, que recibirá por argumento en sus constructor las dimensiones de la matriz n, m, es importante usar el concepto de matriz dinamica en la implementacion del constructor, asi mismo liberar memoria en el destructor.

CMatrix
- int **p - int m,n
- CMatrix(const int &m,const int &n) - CMatrix(CMatrix &M) - CMatrix transpose() - CMatrix sign_matrix() - void print_matrix() - ∼CMatrix()

- Generar una matrix dinamica haciendo uso de **new**[] en el constructor de una clase, dado un n, m dimensiones de la matriz y con valores aleatorios entre -50 y 50.
- Implementar un metodo que nos genere una nueva matriz que resulta ser la transpuesta a la matriz actual.
- Implementar el metodo que genere una sign matrix, para valores positivos se usa 1, negativos -1 y el cero se mantiene.
- Implementar el metodo que muestra la matriz separados por un espacio cada elemento de la fila.
- Por ultimo implementar el destructor que se encargar de liberar memoria haciendo uso de **delete** []

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`)