Caso #1, 5%
Topic: #gettingintoC
Due date: 08-24-2021, 9:00pm
Send to: epinedatec@gmail.com
Subject: Estructuras - Caso #1
Groups of: 1

Description
Proceda a programar en C++ los siguientes ejercicios:

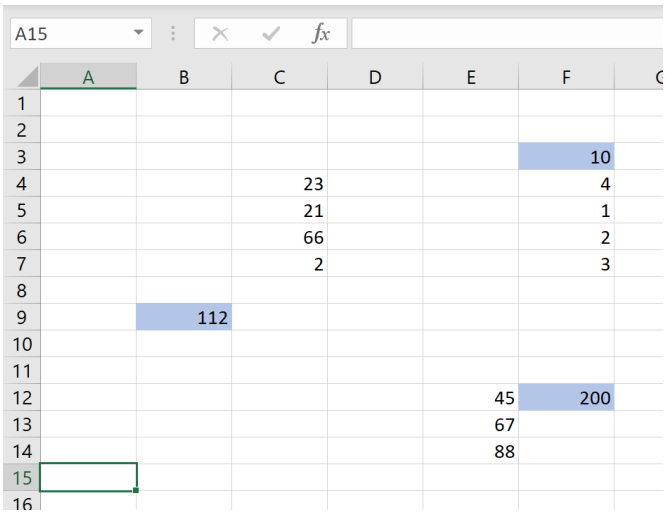
1. Dado un arreglo de strings retornar subgrupos de strings que coinciden entre ellas en dos palabras cualesquiera sin repetición y sacando todas las combinaciones posibles, indicando las dos palabras de match

ejemplo

Input: [‘estoy escribiendo la tarea’, ‘mañana voy en carro’, ‘estoy en el carro escribiendo la poesía’, ‘voy mañana a ver la tarea’]

Output:
estoy escribiendo la tarea, voy mañana a ver la tarea -> tarea la
mañana voy en carro, estoy en el carro escribiendo la poseía -> carro en
estoy escribiendo la tarea, estoy en el carro escribiendo la poseía -> estoy la
estoy escribiendo la tarea, estoy en el carro escribiendo la poseía -> escribiendo estoy
estoy escribiendo la tarea, estoy en el carro escribiendo la poseía -> escribiendo la

2. Diseñe por medio de un struct la definición de un spreadsheet (hoja de calculo), debe poder definir valores enteros ubicados en diferentes celdas, también celdas que contienen operaciones de SUM y dichas operaciones indican cuales son las celdas que suma. Cree un programa que evalúe el struct y retorne otro struct que contenga las sumas resueltas.



Caso #2, 5%
Topic: #sorting
Due date: 09-16-2021, 9:00pm
Send to: epinedatec@gmail.com
Subject: Estructuras - Caso #2
Groups of: 1

Description
Proceda a codear en C++ las siguientes funciones:

1. void sortBySelectionSort(float* pValoresAOrdenar, int pCantidadElementos); el cual debe recibir un puntero a un arreglo de números flotantes y proceder a ordenar los números usando el algoritmos de selectionsort.
2. Implemente una lista enlazada simple que permita almacenar en memoria jugadores. Los jugadores nodo de la lista deben implementarse con un struct y deben incluir un número único y un nombre. La lista simple debe implementar las siguientes funciones:
    - isEmpty(), retorna true en caso de estar vacía
    - getQuantity(), retorna la cantidad de elementos existentes en la lista
    - addPlayer(int pNumber, string pNombre), agrega un nuevo jugador al final de la lista, retorna true si logró agregarlo sin errores
    - removePlayer(int pNumber), busca un jugador con ese número y lo remueve de la lista, retorna true si efectivamente lo encontró y lo removió
    - insertPlayer(int pNumber, string pNombre, int pPosition), inserta un nuevo jugador en una posición específica de la lista, siendo el primer elemento la posición 0. Si la posición no se existe se agrega al final.
    - listPlayers(), procede a imprimir la lista de jugadores
3. void sortByInsertionSort(ListaDinamica *pListaJugadores); la función debe recibir un puntero a una lista de jugadores creada en el ejercicio 2. El algoritmo procede a ordenar la lista con el algoritmo de insertionsort usando el nombre del jugador como llave del ordenamiento. string::compare - C++ Reference (cplusplus.com)

El caso debe cumplir:
- respetar las prácticas estudiadas en clean code
- la lista, nodos y los dos algoritmos de ordenamiento deben quedar en archivos .h separados
- el idioma dentro del código fuente debe ser inglés
- cree un programa que tengan un main que permita hacer las pruebas de los 3 ejercicios: crear un arreglo de varios elementos y que se demuestre que efectivamente lo ordena, que se pruebe que funcionan bien las funciones de la lista y lo mismo que dicha lista quede efectivamente bien ordenada
- cualquier sospecha de copia anulará el trabajo, tenga en cuenta que ninguno de los ejercicios amerita hacer uso de código o librerías copiadas de internet
- la tarea deberá entregarla en un correo como un link de github, cualquier explicación adicional la agregan al readme.md y el último commit no debe superar la fecha y hora de entrega

Caso #3, 15%
Topic: #lineardatastructures
Due date: 03-Oct-2021, 11:00pm, last commit on main
Revision: on remote call with the professor
Groups of: 2

Description

Las bodegas en las distribuidoras de productos normalmente son altas columnas de algun producto apilados en paletas. Las distribuidoras en la mañana toman los pedidos que indican todos los productos que deben alistarse. Dicha orden es tomada por un montacarga que se dirige a la columna en la bodega, saca la cantidad de paletas que sea necesario para completar el pedido y vuelve a poner en la columna lo que sobra en la paleta. Si completa el pedido lo marca como terminado y si falta cantidad para cierto producto lo marca como incompleto.

Proceda a diseñar e implementar las estructuras de datos necesarias para poder crear una simulación de su bodega, los pedidos y los montacargas esto es:

- la bodega se especifica indicando cuantas columnas hay de cada tipo de producto, la cantidad de paletas que caben por columna y la cantidad de unidades que caben por paleta. Cada bodega va a ser una pila.
- los pedidos son una lista de pedidos, para cada pedido se especifica un arreglo con los productos y la cantidad de por producto que se está solicitando
- los montacargas se especifican por cantidad de montacargas disponibles y el tiempo en milisegundos que un montacarga dura bajando y subiendo una paleta. Cada montacarga tendrá su propia cola de pedidos.

Su programa inicialmente debe recorrer la lista de pedidos y aleatoriamente distribuir todos los pedidos entre los montacargas, metiendo cada orden en la cola respectiva de cada montacarga. Seguidamente en un bucle cada montacarga saca el pedido a procesar, recorriendo los productos que debe completar, selecciona la pila respectiva y procede a sacar las paletas de la pila para completar la cantidad solicitada en el pedido de cada producto, cada montacarga dura subiendo y bajando las paletas (push y pop) el tiempo en milisegundos especificado. Si no hay suficiente producto se procede con el siguiente producto hasta terminarlos y se marca el pedido como incompleto, de lo contrario queda en estado completo.

Una vez que hayan terminado todos los montacargas (todos los pedidos), se imprime en pantalla para cada pedido su estado y la cantidad de milisegundos que se demoró en completarse.

Detalles de implementación

- todas las listas, pilas y colas deben ser implementadas por el estudiante
- no es necesario pedir información al usuario, las ordenes y la configuración se definen en el main y con un .h de constantes
- separe cada definición de estructura y lógicas en archivos separados .cpp y .h
- no es necesario utilizar hilos de ejecución
- cualquier sospecha de copia anulará el trabajo
- si copia código de internet de algun tipo escriba a modo comentario el url de donde lo sacó, eso no le exime de entender el código y dominarlo, sin embargo, por la naturaleza del problema no se visualiza que sea necesario utilizar nada que no puedan programar
- deberá seguir las prácticas estudiadas en clean code
- el código fuente para ser revisado deberá hacerle commit al main branch en github a la hora límite de finalización. Se recomienda que también haga uso de un branch de develop donde los estudiantes puedan hacer merge del código y que haya un branch para cada integrante del grupo de trabajo