Segundo Parcial de Algoritmos y Estructura de Datos – **Consideraciones**

Estimados,

el examen consta del desarrollo de 3 algoritmos cortos, para desarrollar en general forma modular y aprovechando **funciones de biblioteca disponibles e incluso sugeridas para cada caso**.

Los ejercicios involucran conceptos básicos de manejo de archivos, memoria estática y estructuras enlazadas, no son complejos. Donde la estructura de datos es más simple (por ej sólo vector), se pide algo más a desarrollar, de manera que todos los ejercicios queden de complejidad similar.

Cada ejercicio tiene un tiempo estimado indicado y son ejercicios cortos tanto en enunciado como en desarrollo no excediendo las 10-15 líneas. Y son ejercicios conceptuales, con lo cual ***es necesario poder desarrollar cada uno de ellos en forma razonable para poder aprobar* *la materia***, y muy bien para obtener nota de promoción. En uno de los ejercicios puede elegir incluso el algoritmo a desarrollar entre dos opciones que se indican claramente.

Y en cada caso ya se dan los tipos de datos disponibles y hasta un diagrama de las estructuras para facilitar visualización.

Teniendo en cuenta esto, sepa que los ejercicios se despliegan en forma aleatoria, indican el tiempo estimado y hasta una cota por exceso. TOME UN TIEMPO PARA REVISAR ANTES DE PASAR AL EJERCICIO SIGUIENTE, YA QUE EL COMPORTAMIENTO DE LOS EJERCICIOS ES SECUENCIAL **(SIN PODER VOLVER PARA ATRÁS)**.

Recuerde que los ejercicios no necesitan “compilar” ya que son partes de programas, ni siquiera precisan ser desarrollados en código, sino que pueden ser desarrollados en un editor de texto o en una hoja, tanto a nivel código c++ como a nivel diagrama.

***Por favor recuerde realizar las consultas a través del chat y no en forma oral para no distraer al resto.***

Suerte y a demostrar sus conocimientos adquiridos en la materia!

**-------------------------------------------------------------------------------------------------------------------------------------------**

**Desarrollo de Algoritmo – 3 puntos** - **Tiempo estimado 20-25 min**

|  |  |
| --- | --- |
| Una agencia de viajes desea llevar estadísticas de los destinos turísticos más consultados.  Para ello cuenta con un vector de Destinos **VDest** ya cargado con n destinos (máx 50) y con la siguiente estructura: | |
| struct tDest {  int CodDest;  int Categ;  int CC; }; (Cant de Consultas realizadas)    tDest VDest[50]; | Visualmente   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | CodDest | Categ | CC |  |  |  |  |  |  |  | | 0 |  |  |  |  |  |  |  | |  |  | | 1 |  |  |  |  |  |  | |  |  | | .. |  |  |  |  |  |  |  |  |  |  | | 49 |  |  |  |  |  |  |  |  |  |  |   **VDest** |
| Se pide:  Desarrolle un módulo void **BuscAgrDestino** que recibe  el vector **VDest**,  la cantidad de destinos cargados **n** y  un cod de Destino a buscar **DestABuscar**.  Agregue los parámetros que considere necesarios.  Debe **buscar dicho Destino** en el vector.  Si lo encuentra, debe incrementar en uno la Cantidad de Consultas realizadas.  **Si no lo encuentra, debe agregar** dicho destino al vector (en Categoría como no la tiene, se la debe solicitar al usuario). No hace falta controlar si el vector se llenó y no tiene lugar.  Debe **devolver por referencia**, la Categoría correspondiente al destino buscado sea que lo agregó o lo encontró.  *Dispone del módulo estándar (que puede aplicar sin desarrollar)*  *int* ***buscar****(tvec vec[], int n, int datoabuscar)*  *que busca secuencialmente un elemento en un vector retornando el índice donde lo encuentra, si el dato buscado está o el valor -1 si lo buscado no está en el vector. Ya busca por el campo que hace falta, no hace falta indicar nada más que el datoabuscar.* | |

Posible Solución

void **BuscAgrDestino** (tDest VDest, int & n, DestABuscar, int & Categ ) {

int pos=**buscar**(VDest,n,DestABuscar)**;**

if(pos!=-1){

VDest[pos].CC++;

Categ = VDest[pos].Categ;

} else {

VDest[n].CodDest =DestABuscar;

cin >> Categ;

VDest[n].Categ = Categ;

VDest[n].CC=1;

n++;

};

};

-------------------------------------------------------------------------------------------------------------------------------------------

**Desarrollo de Algoritmo - 4 puntos** - **Tiempo estimado 20-25 min**

|  |  |
| --- | --- |
| Desarrolle la parte de un main que dados un archivo **fVta** y una lista **LVta** con la misma estructura, detallada abajo: | |
| struct tVta {  int CodP;  float Fact; (facturación de la venta)  };  struct Nodo {  tVta INFO;  Nodo\* sig;  }; | Visualmente   |  |  |  |  |  | | --- | --- | --- | --- | --- | | LVta |  | INFO | | Sig | |  | CodP | Fact | |
| -**lea un registro** de un archivo fVta que ya está abierto para lectura,  -**busque dicha venta** (por doble campo, coincidencia del CodP y de la Fact) en la lista LVta ya cargada sin orden (no se preocupe no hay dos nodos con la misma combinación de CodP y Fact),  -en caso en que no lo encuentra, **lo inserte al final**,  -**cargue en una variable** de tipo punterola dirección del nodo (en que lo encontró o agregó a la lista).  No coloque la cabecera “main {” ya que esto es una parte de un pgm que seguro arrancó antes.  *El módulo estándar de la biblioteca* ***buscar*** *en una lista,**no le sirve ya que ese busca por un campo y acá debe buscar coincidencia por doble campo. Puede desarrollar un módulo* ***buscardoble*** *que busca como necesita por doble campo, o puede poner directo las sentencias en el main sin desarrollar módulo que es más corto.*  *Dispone además del módulo estándar (que puede aplicar sin desarrollar)*  *Nodo\** ***insertarAlFinal****(Nodo\*& LVta, int RVta)*  *Sin embargo,* ***preferentemente no lo utilice*** *ya que como ya recorrió la lista para buscar, sería poco óptimo volver a hacerlo.  Si no encuentra otra opción, puede utilizarlo pero no obtendrá el máximo puntaje para este ejercicio.* | |

Posible Solución

**fread** (&RVta, sizeof (RVta), 1, fVta); // o **read**(RVTA,fVta)

Nodo \* ptr = Lista;

Nodo \* ant=NULL;

**while** (ptr!=NULL && !(ptr->INFO.CodP==RVta.CodP && ptr->INFO.Fact==RVta.Fact)) {

ant=ptr;

ptr = ptr-> sig;

};

If (ptr ==NULL) {

ptr = new Nodo();

ptr-> INFO = RVta;

ptr-> sig = NULL;

ant-> sig = ptr;

}

Otra manera de buscar doble (pero es más larga) podría ser ir recorriendo la lista y cuando encuentro la doble coincidencia, me guardo en otro puntero la pos encontrada. Y sino ese puntero queda con NULL.

Y para hacer más eficiente, cuando lo encuentra al puntero de la recorrida lo pongo en NULL para que no siga buscando.

**fread** (&RVta, sizeof (RVta), 1, fVta); // o **read**(RVTA,fVta)

Nodo \* ptr = Lista;

Nodo \* ant=NULL;

Nodo\* ptrencontr=NULL;

**while** (ptr!=NULL) {

if (ptr->INFO.CodP==RVta.CodP && ptr->INFO.Fact==RVta.Fact)) {

ptrencontr=ptr;

ptr=NULL; // para que deje de buscar

} else {

ant=ptr;

ptr = ptr-> sig;

};

};

Y luego el if al salir es

if (ptrencontr==NULL) // en vez de if (ptr==NULL)

// resto igual

ptr = new Nodo();

ptr-> INFO = RVta;

ptr-> sig = NULL;

ant-> sig = ptr;

}

//y en este caso se agrega un else

// para que si lo encontró, lo deje en ptr como en el caso de no haberlo encontrado

else {

ptr = ptrencontr;

}

-------------------------------------------------------------------------------------------------------------------------------------------

**Desarrollo de Algoritmo - 3 puntos** - **Tiempo estimado 20-25 min**

|  |  |
| --- | --- |
| Una compañía eléctrica de una ciudad cuenta con un vector de zonas ya cargado con n zonas (máx 100) con la estructura detallada abajo, y listas de clientes por cada zona ya cargadas, con la estructura detallada abajo: | |
| struct tZona {  int Zona;  float FactTot; (facturación total de la zona)  nodo \* LCli (Lista de Clientes de la zona)  };  tZona VZonas[100];  struct Nodo {  int DNI;  Nodo\* sig;  }; | Visualmente   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | Zona | FactTot | LCli |  |  |  |  | | 1 |  |  |  |  | DNI | sig | | .. |  |  |  |  |  |  |  | | 99 |  |  |  |  |  |  |  |   **VZonas** |
| Se pide desarrollar ***UNO de los dos algoritmos siguientes (puede elegir el que prefiera, aunque la opción b llega a valer a lo sumo 2 de los 3 puntos)***:  **a)** módulo void **AsignarCliente** que recibe los siguientes parámetros:  el vector **VZonas**, la cantidad de elementos cargados **n**, una Zona de tipo entero a buscar **ZonaABuscar,** un nro de documento **DNI** y facturación **Fact** de un cliente.  Debe buscar dicha Zona en el vector (está seguro), acumula la facturación total de la Zona e inserta ordenado dicho cliente en la lista correspondiente.  *(la solución es muy corta)*  *Dispone de los módulos estándar (que puede aplicar sin desarrollar)*  *int* ***buscar****(tvec vec[], int n, int datoabuscar)*  *que busca secuencialmente un elemento en un vector retornando el índice donde lo encuentra, si el dato buscado está o el valor -1 si lo buscado no está en el vector*  *Nodo\** ***insertarOrdenado****(Nodo & Lista, int infodelnodo)*  *inserta creciente en una lista según el criterio de ordenamiento requerido*  **b)** el módulo void **ListadoClientes**, que recibe los siguientes parámetros:  el vector **VZonas** y la cantidad de elementos cargados **n,** e imprime el siguiente listado ***sin liberar*** *los nodos*:  Zona: 999 Fact Tot: 9999.99  DNI  99.999.999  99.999.999  … | |

Posible Solución

void **AsignarCliente** (tZona VZonas, int n, int ZonaABuscar, int DNI, float Fact){

int pos=**buscar**(VZona,n, ZonaABuscar)**;**

VZonas[pos].FacTot+=Fact;

Noto\* ptr = **insertarOrdenado**(VZonas[pos].LCli,DNI);

}

void **ListadoClientes** (tZona VZonas, int n){

Nodo\* ptr;

for (i=0,i<n,i++) {

cout << “Zona “ << VZonas[i].Zona << “Fact Tot “ << VZonas[i].FactTot << endl;

ptr = VZonas[i].LCli

while (ptr!=NULL) {

cout << ptr -> DNI << endl;

ptr = ptr -> sig;

};

};

};

-------------------------------------------------------------------------------------------------------------------------------------------