Ejercicio de Práctica en Papel

Consigna:

El sistema de becas de la UTN-FRSF permite asignar alumnos (futuros becarios) a proyectos de investigación. Un PROYECTO DE INVESTIGACIÓN queda definido por un nombre (string), código (entero), apellido del director (string), departamento (codificación con valor entero: 0=CIVIL, 1=ELECTRICA, 2=INDUSTRIAL, 3=SISTEMAS, 4=MECANICA) y cantidad de becarios solicitados (valor entero <= 2). A su vez, un ALUMNO queda definido de la siguiente manera:

```
struct Alumno {

      string nombre; //nombre y

      apellidoint dni; //número

      de documento

      int aprobadas; //cantidad de materias aprobadas

        int carrera; //ingeniería - igual codificación que los
                              PROYECTOS.

};
```

Para la asignación de becarios, el sistema posee dos listas: INSCRIPTOS (lista de alumnos interesados en ser becarios) y PROYECTOS (lista de proyectos). La lista de proyectos se encuentra ordenada por prioridad. Esto quiere decir que los proyectos ubicados en las primeras posiciones tienen mayor prioridad que los ubicados en las siguientes posiciones. Por otra parte, la lista de INSCRIPTOS no tiene ningún orden en particular. Ambas listas se encuentran implementadas en estructuras estáticas (tamaño máximo de INSCRIPTOS 200 y de PROYECTOS 30).

En base a estas definiciones, se le solicita:

a) Defina los tipos de datos y las estructuras necesarias para representar en C++ la información que ya se dispone en el sistema de becas.

b) Implemente la función `proyectosISI` en base al siguiente prototipo:

```
                    ListaProyectos
          proyectosISI(ListaProyectos p);
```

Esta función recibe la lista de PROYECTOS y devuelve una sublista que contiene únicamente los proyectos del Departamento Sistemas ordenados según su prioridad.

c) Sabiendo que dispone de las siguientes estructuras:

```
struct Asignacion {

      int dni_estudiante; //documento del alumno asignado

      int cod_proyecto; //codigo del proyecto al cual se asignó el alumno.

};

struct NodoAsignacion {

      Asignacion asig;

      NodoAsignacion* sig;

};
```

Implemente la función *asignarBecarios* que recibe las listas de inscriptos junto con la lista generada en el ítem b). La función retorna una lista dinámica que contiene las asignaciones de alumnos a proyectos siguiendo el criterio "*los proyectos de mayor prioridad reciben como becarios a los inscriptos de la carrera con mayor cantidad de materias aprobadas*". Considerar que una vez que un alumno ha sido asignado a un proyecto, no puede asignarse a otro. Ante alumnos con igual cantidad de materias rendidas, se elige aleatoriamente la asignación.

d) Describa en los siguientes renglones el objetivo de la función que se presenta más abajo, y optener la función O.

.....

.....

.....

.....

.....

```
void funcionItemD(Alumno a) {

      if(a.dni<=0) return;

      cout<<a.dni % 10;

      a.dni /= 10;

      funcionItemD(a);

}
```

e) Defina e implemente una función de complejidad log N que, recibiendo como argumento la lista devuelta en el item b y un valor entero N, determine si existe un proyecto con prioridad N en la lista. Justifique porque su algoritmo tiene la complejidad solicitada.

Importante: Para la resolución del problema el alumno puede codificar todas las estructuras de datos y funciones que considere necesarias. Los campos de las estructuras deben respetar lo enunciado en la consigna. Los parámetros formales de las funciones (cantidad y tipo) deben definirse según los objetivos propuestos. El puntaje final obtenido tendrá en cuenta la eficiencia de la estrategia de resolución elegida.

Ejercicio de Práctica en Papel

Al programar en C++, el CÓDIGO FUENTE puede ser visto como un conjunto de LINEAS. Con un alto nivel de abstracción, una LINEA queda definida por su número (valor entero que indica su orden en el conjunto) y una sentencia expresada en C++ (texto sin longitud fija).
Desde las cátedras de programación, podemos considerar que cuando planteamos un problema, cada ESTUDIANTE realiza su ENTREGA de CODIGO FUENTE. Entonces, una ENTREGA queda definida por un identificador de estudiante (valor entero) y un CODIGO FUENTE. A su vez, una COMISIÓN queda definida por su nombre (tipo char) y por el conjunto de todas las entregas asociadas a sus estudiantes (máximo 250).

En base a estas definiciones, se le solicita:

a) Defina todas las estructuras de datos necesarias para representar una COMISIÓN. Luego, declare las variables que considere necesarias para instanciar las 3 COMISIONES (A, B y C) de AEDD. Se sabe que los estudiantes asociados a las comisiones A, B y C son 123, 145 y 198, respectivamente.

Considerando que las entregas de los estudiantes ya se encuentran cargadas en las variables declaradas, se le solicita:

b) Defina e implemente la función codigosDuplicados la cual recibe las 3 COMISIONES y devuelve un listado que contiene los identificadores de estudiantes que han entregado CODIGOS FUENTE duplicados. Se considera que dos CODIGOS FUENTE son duplicados si sus LINEAS son exactamente iguales (tanto en su número como en su contenido textual).

c) Defina e implemente la función ordenarLineas que recibe un CODIGO FUENTE con sus líneas desordenadas. Se considera que un CODIGO FUENTE tiene sus líneas desordenadas si el número de LINEA no se corresponde con el orden en el cual dicha LINEA ha sido registrada dentro del conjunto de líneas. Como resultado de la ejecución de la función, el código recibido debe quedar ordenado. Justifique la complejidad que su algoritmo tiene.

d) Defina e implemente la función cantidadDeLineas que recibe un CODIGO FUENTE y devuelve la cantidad de líneas que posee.

Importante: Para la resolución del problema el alumno puede codificar todas las funciones que considere necesarias. No se admitirán soluciones que incorporen campos redundantes a las estructuras de datos planteadas. El puntaje final obtenido tendrá en cuenta la eficiencia de la estrategia de resolución elegida.

Consigna:
El sistema de software utilizado por el correo para la mensajería e-commerce mantiene la información de los **envíos pendientes de entrega** en una lista dinámica definida en base a la siguientes estructura de datos:

```
struct Envio {

   int numero; //numero de seguimiento.

   Paquete paq; //información del paquete a ser entregado.

   Informacion origen, destino; //información del remitente y destinatario.

};
```

Se sabe que para cada elemento de tipo `Paquete`, se considera relevante su tamaño (ancho, largo y alto, todos en cm), si el contenido es frágil o no (valor booleano) y el peso (en Kg). Por otro lado, para los campos de tipo `Informacion` se almacena nombre y apellido (tipo String), DNI (tipo entero) y `Dirección` (calle, número, ciudad y provincia).

Tomando como base estas definiciones, se solicita:

a) Definí todas las estructuras de datos que consideres necesarias para implementar la lista de **envíos pendientes de entrega**. Haciendo uso de tus definiciones, crea una lista vacía local a la función `main()`.

b) Cada cartero a cargo de los `Envios`, reparte `Paquetes` de un mismo tipo. Para esto, los `Paquetes` en lista son clasificados según su tamaño en Grandes, Medianos, o Pequeños. La siguiente tabla muestra la clasificación de los paquetes según su volumen:

Clase	Volumen (cm^3)
G	>= 100000
M	entre 15001 y 100000
P	<= 15000

Definí e implementá una función que, tomando como argumento la lista de envíos pendientes y una clase de paquetes (definida como char), genere y devuelva la sublista de `Envios` que contienen `Paquetes` de la clase indicada. Es decir, la lista que se devuelve la función indica los `Envios` que el cartero debe repartir según la clase indicada (`LISTA1`). Los `Envios` incorporados en la sublista, no deben removerse de la lista original.

c) A medida que el cartero realiza el reparto, el contenido de `LISTA1` es actualizado. Al realizar una entrega, el cartero hace uso de la función `eliminarEnvio()` para dicha actualización. La función `eliminarEnvio()` recibe la lista de envíos pendientes de entregar por el cartero y el número de seguimiento del `Envio` entregado. Como resultado de ejecutar esta función, la lista de envíos pendientes se actualiza removiendo el `Envio` entregado. Justifique la complejidad que su algoritmo tiene.

d) Cuando el cartero regresa a la central (luego del reparto), aún pueden quedan `Envios` en `LISTA1`, ya que no siempre los destinatarios están en casa. Definí e implementá una función `enviosNoEntregados()` que reciba como parámetro la lista resultante luego del reparto y retorne la cantidad de envíos no entregados. Esta función deberá ser implementada de forma recursiva.

e) Los números de seguimiento se dan en secuencia una vez que el `Envio` ingresa al sistema. Sin embargo, cuando los `Envios` arriban a la sucursal de destino es frecuente que dicho orden se pierda (ya que los `Paquetes` pueden llegar a destiempo). Se te solicita que elabores una función que determine, si en un momento dado, se encuentra al menos una secuencia de 30 números de seguimiento consecutivos crecientes (uno en uno) en la lista de envíos pendientes de entrega. Para esto, podrás hacer uso de estructuras de datos auxiliares.

Importante: Para la resolución del problema podes codificar todas las funciones que consideres necesarias. Los campos de las estructuras de datos deben respetar lo enunciado en la consigna. En los casos donde no se indica un prototipo explícito en la consigna, los parámetros formales de funciones (cantidad y tipo) deben definirse según los objetivos propuestos. El puntaje final obtenido tendrá en cuenta la eficiencia de la estrategia de resolución elegida.

Consigna:

Las INSCRIPCIONES a un evento deportivo se registran cronológicamente en una lista. Como máximo, en un evento se registran 200 inscripciones. Cada INSCRIPCION consiste en la INFORMACION PERSONAL del competidor y el conjunto de DISCIPLINAS a las cuales desea inscribirse. La INFORMACION PERSONAL contiene nombre y apellido (tipo string), edad (valor entero), dni (arreglo de 8 valores enteros), sexo ('f' o 'm') y apto físico (valor booleano utilizado por la organización para indicar que el competidor es apto para las disciplinas elegidas). Un competidor puede anotarse en hasta 3 DISCIPLINAS. Las DISCIPLINAS existentes son: 0 = futbol, 1 = basquet, 2 = voley, 3 = ciclismo, 4 = vela, 5 = padel, 6 = tenis y 7 = handball.

Teniendo en cuenta la descripción previa, se le solicita:

a) Defina los tipos de datos y las estructuras necesarias para representar las INSCRIPCIONES a los JUEGOS DEPORTIVOS UTN. Para el manejo de la INFORMACION PERSONAL, haga uso de un tipo de dato abstracto.

b) Indique la/las sentencia/sentencias que utilizaría dentro de *main* para declarar e inicializar los datos personales de su propia inscripción.

c) Defina e implemente la función *inscriptosEnDisciplina* que recibe una lista de INSCRIPCIONES, el identificador de una disciplina y el sexo de los competidores. La función debe retornar una lista con la INFORMACION PERSONAL de todos los competidores del sexo recibido como argumento que se han anotado a la disciplina indicada.

f) Defina e implemente la función recursiva *verEnInverso* que recibe una lista de INSCRIPCIONES y muestra por pantalla la INFORMACION PERSONAL de todos los inscriptos pero en orden inverso (es decir, desde la INSCRIPCION mas reciente a la más antigua). Justifique la complejidad que su algoritmo tiene.

d) Defina e implemente la función *cancelarInscripción* que recibe una lista de INSCRIPCIONES junto con el dni de un competidor. La función debe eliminar la INSCRIPCIÓN asociada a dicho competidor del listado.

Importante: Para la resolución del problema el alumno puede codificar todas las funciones que considere necesarias. La cantidad de parámetros formales asociados a cada función debe ser definida por el alumno. El puntaje final obtenido tendrá en cuenta la eficiencia de la estrategia de resolución elegida.