Programación 3

Trabajo obligatorio

Curso nocturno 2018

Licenciatura en Informática

Tutor:	**Ron, Ariel.**	
Integrantes:	**Chocho, Washington**	**- 4.548.794-9**
	Pías, Richard	**- 1.924.591-2**
	Segovia, Joaquín	**- 4.739.544-4**
	Torres, Mathias	**- 4.223.291-4**

Tabla de contenido

1) Grafo de previaturas

Para este trabajo de laboratorio, una de las tareas principales es el diseño de las estructuras, entre las cuales se encuentra el grafo de previaturas de las asignaturas.

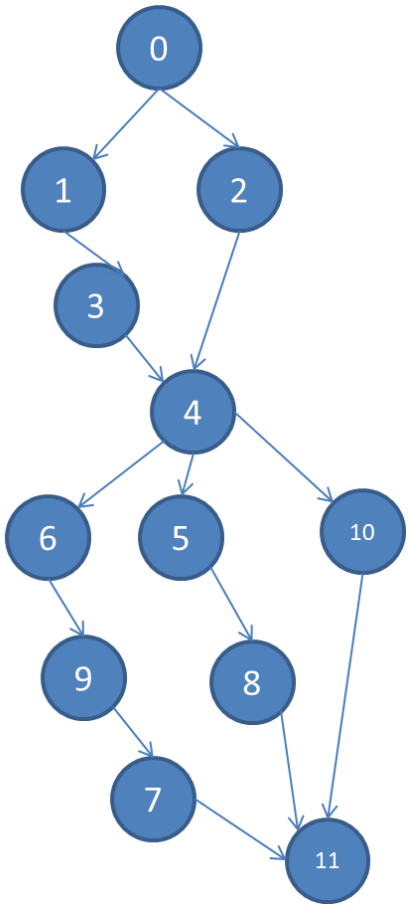
Donde los vértices representan las asignaturas y las aristas representaran la previaturas entre dos materias.

Dicho grafo, es conexo, ya que todos los nodos están unidos a la estructura en si, además es un grafo dirigido pues las flechas "van" en un sentido. Es simple pues no presenta lazos, ya que una materia no puede tenerse como previa a sí misma. No es completo ya que no todas las materias se conectarían con todas las restantes directamente.

Por otro lado, el grafo es acíclico ya que no presenta ciclos, pues sino sería imposible comenzar alguna asignatura.

Ejemplo:

ID	Nombre Asignatura	
0	**Introducción a la Gastronomía**	**IG**
1	Cocina y Gastronomía 1	**CG1**
2	Nutrición y Productos 1	**NP1**
3	Cocina y Gastronomía 2	**CG2**
4	**Proyectos Gastronómicos 1**	**PG1**
5	Comercialización 1	**C1**
6	Cocina y Gastronomía 3	**CG3**
7	Enología y Bebidas 1	**EB1**
8	Comercialización 2	**C2**
9	Cocina y gastronomía 4	**CG4**
10	Nutrición y Productos 2	**NP2**
11	**Proyectos Gastronómicos 2**	**PG2**



2) Análisis: tipos de datos abstractos (TAD)

Se presenta a continuación, se presentan el resultado del análisis de los datos de tipos abstractos (TAD). Para esto se tuvieron en cuenta los requerimientos.

1. Alumno = Producto Cartesiano (CI, Nombre, Apellido, Domicilio, Telefono, Escolaridades)
2. Asignatura = Producto Cartesiano (ID, Nombre)
3. Escolaridad = Producto Cartesiano (ID, Fecha, Calificacion)
4. Alumnos = Diccionario (Alumno)
5. Asignaturas = Diccionario (Asignatura)
6. Escolaridades = Diccionario (Escolaridad)
7. Previaturas = Grafo (Asignatura)
8. CI = Entero
9. Nombre = String
10. Apellido = String
11. Domicilio = String
12. Telefono = Entero
13. ID = Entero
14. Fecha = Fecha
15. Calificacion = Entero

Los siguientes puntos corresponden a la justificación.

1. La entidad alumno, se representara como un producto cartesiano donde se guardaran todos los datos referentes a él tales como CI, siendo esta su identificatorio, su nombre, apellido, domicilio, teléfono y sus aprobaciones.
2. La entidad asignatura, se definirá como un producto cartesiano para guardar los datos tales como nombre de asignatura y numero de asignatura el cual es su clave identificatoria.
3. La entidad escolaridad, la cual denominamos como sinónimo de aprobación, presentada como un producto cartesiano, contendrá el número identificador de asignatura, además de la fecha de aprobación y la calificación con la que se aprobó.
4. La colección de alumnos se define como un diccionario ya que los alumnos son identificados por su C.I. y son elementos que no se repiten.
5. La colección de asignaturas se especifica como un diccionario puesto que se identifican por su número de asignatura y no se pueden repetir.
6. La colección de escolaridades, se expresa como un diccionario ya que se identificaran por el número identificador de asignatura, los cuales no pueden ser repetidos ya que un alumno no puede aprobar dos veces la misma asignatura.
7. Por último, la colección previaturas, se representa como un grafo, ya que esta estructura nos permite modelar correctamente la realidad del sistema de previaturas, donde tenemos asignaturas y las relaciones (previas) entre ellas.

3) Estructuras de datos para representar los TAD

Alumno, Asignatura, Escolaridad: En estos tres casos, elegimos tipos estructurados (struct) de C++, dado que nos interesa almacenar características que no necesariamente son del mismo tipo.

Diccionario de Alumnos: La colección de alumnos se define como un diccionario ya que los alumnos son identificados por su C.I. y son elementos que no se repiten. La estructura elegida para su representación es la de Árbol Binario de Búsqueda (ABB) ya que nos permite guardarlos ordenadamente por la C.I. de cada alumno, y así poder listarlos por orden, ya que un requerimiento nos pide que listemos los alumnos por su CI de menor a mayor.

Diccionario de Asignaturas: La colección de asignaturas se especifica como un diccionario puesto que se identifican por su número de asignatura. En este caso se optó por utilizar un arreglo "simple" con la constante n definida por los desarrolladores ya que la cantidad de materias es constante y finita. Como el ingreso de asignaturas se realiza ordenadamente y los números identificatorio se insertan consecutivamente, podemos utilizar el arreglo para el requerimiento de listar las asignaturas de modo ordenado por su ID de menor a mayor.

Diccionario de Escolaridades: La colección de escolaridades, se expresa como un diccionario ya que se identificaran por el número identificador de asignatura, los cuales no pueden ser repetidos. Esta estructura será ordenada por fecha de aprobación. Dicha estructura estará contenida en cada alumno en forma lista (simple), y contendrá todas las llamadas escolaridades.

Grafo de Previaturas: Por último, la colección previaturas, se representa como un grafo. Se optó por una lista de adyacencia ya que nos pareció la forma más correcta ya que hay muy pocas aristas así poder ocupar menos memoria.

4) Diagrama de jerarquía de módulos

Se adjuntará a continuación el diagrama de jerarquía de módulos con las inclusiones.

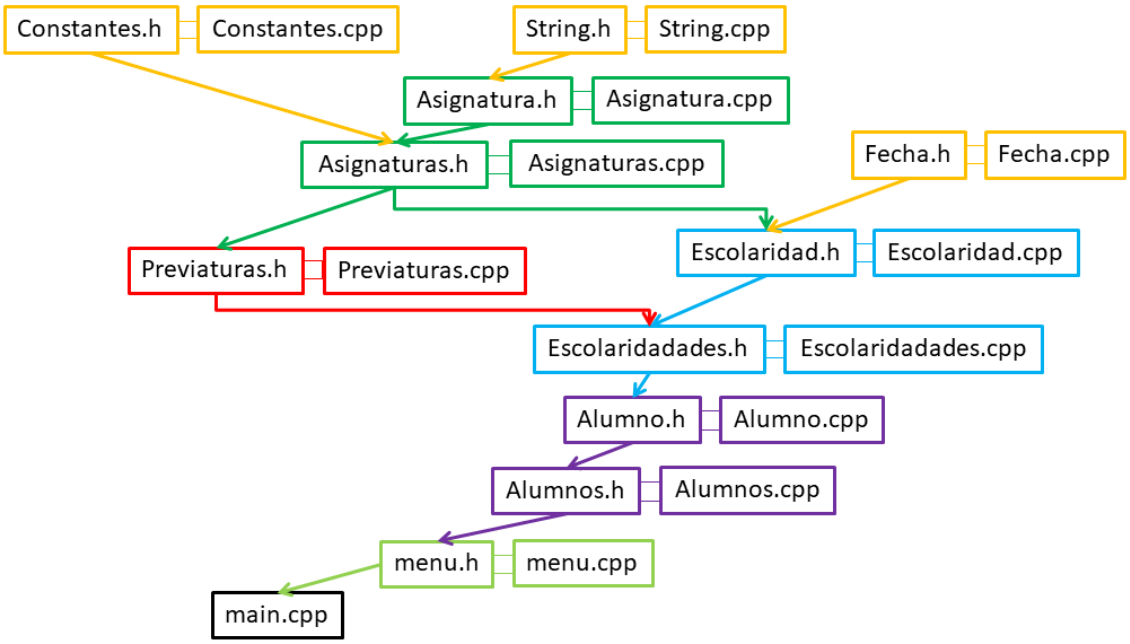
Como primera instancia tenemos al módulo 'String' el cual le hereda a 'Asignatura' y 'Constantes' le hereda a 'Asignaturas'.

El módulo 'Asignatura' le hereda a 'Asignaturas', la cual contendrá la colección de asignaturas; además esta última le hereda al módulo 'Previaturas' el que guardara las previaturas (id de asignatura) de cada asignatura, en el caso que tuviera.

'Escolaridad' (una aprobación) hereda de 'Fecha' ya que el requerimiento nos pide guardar la fecha de aprobación de una materia, además hereda también 'Asignaturas', tambien 'Asignaturas' le hereda a 'Escolaridad'. 'Previaturas' le hereda a 'Escolaridades'.

Justamente 'Alumno', hereda 'Escolaridades' ya que el struct alumno contendrá la escolaridad del mismo. Además, 'Alumnos' hereda de 'Alumno' ya que es una colección que los contendrá.

Por último, 'menu' hereda de 'Alumnos' y le hereda a 'main'. Cabe destacar que el modulo menú contendrá los textos y lógica que se mostraran dinámicamente dependiendo si las asignaturas están cargadas o no.



*se adjunta en el .ZIP la imagen en pdf y png para una mejor vizualizacion

5) Encabezados de las primitivas y otras operaciones de cada uno de los TAD

Se exponen en esta sección, los cabezales que se encontraran en los archivos con extensión .h, tales como selectoras y primitivas para los TAD.

Entidad o colección	TAD	
Alumno	**Producto Cartesiano**	
void CargaAlumno(Alumno &a, int ci, String nombre, String apellido, String domicilio, int telefono);		
bool AreIgualesAlumno(Alumno a1, Alumno a2);		
bool MenorQueAlumno(Alumno a1, Alumno a2);		
int DarCIAlumno(Alumno a);		
void DarNombreAlumno(Alumno a, String &nombre);		
void DarApellidoAlumno(Alumno a, String &apellido);		
void DarDomicilioAlumno(Alumno a, String &domicilio);		
int DarTelefonoAlumno(Alumno a);		
void MostrarDatosAlumno(Alumno a);		
Escolaridades DarAprobacionesAlumno(Alumno a);		
void DeleteAlumno(Alumno &a);		
Asignatura	**Producto Cartesiano**	
void CargarAdignatura(Asignatura &a, int id, String nombre);		
void MostrarDatosAsignatura(Asignatura a);		
int DarIdAsignatura(Asignatura a);		
String DarNombreAsignatura(Asignatura a);		
void DeleteAsignatura(Asignatura &a);		
Escolaridad	**Producto Cartesiano**	
void CargarEscolaridad(Escolaridad &e, int idAsignatura, Fecha f, float calificacion);		
void MostarDatosEscolaridad(Escolaridad e);		
int DarIdAprobacion(Escolaridad e);		
Fecha DarFechaAprobacion(Escolaridad e);		
float DarCalificacionAprobacion(Escolaridad e);		
Alumnos	**Diccionario**	
void makeAlumnos(Alumnos &Alumnos);		
bool memberAlumnos(Alumnos alu, Alumno a);		
void insertAlumnos(Alumnos &Alumnos, Alumno a);		
Alumno findAlumnos(Alumnos abb, int ci);		
void modifyAlumnos(Alumnos &abb, Alumno a);		
void deleteAlumnos(Alumnos &abbAlumnos, int ci);		
bool IsVacioAlumnos(Alumnos abbAlumnos);		
Asignaturas	**Diccionario**	
void makeAsignaturas(Asignaturas &a);		
bool memberIDAsignaturas(Asignaturas a, int id);		
bool memberNombreAsignaturas(Asignaturas a, String nombre);		
void insertAsignaturas(Asignaturas &a, Asignatura as);		
Asignatura findAsignaturas(Asignaturas a, int id);		
void modifyAsignaturas(Asignaturas &a, Asignatura as);		
void deleteAsignaturas(Asignaturas &a, int id);		

	Escolaridades	**Diccionario**	
	void makeEscolaridades(Escolaridades &le);		
	bool memberEscolaridades(Escolaridades le, int id);		
	void insertEscolaridades(Escolaridades &le, Escolaridad e);		
	Escolaridad findEscolaridades(Escolaridades &le, int id);		
	void modifyEscolaridades(Escolaridades &le, Escolaridad e);		
	void deleteEscolaridades(Escolaridades &le, int id);		
	void listarAllEscolaridades(Escolaridades esc);		
	void ordenarAllEscolaridades(Escolaridades &escs);		
	bool tieneAlgunaEscolaridad(Escolaridades escs);		
	Previaturas	**Grafo**	
	void makePreviaturas(Previaturas &lp);		
	bool memberPreviaturas(Previaturas lp, int id);		
	void insertPreviaturas(Previaturas &lp, Asignatura as);		
	Asignatura findPreviaturas(Previaturas &lp, int id);		
	void modifyPreviaturas(Previaturas &lp, Asignatura as);		
	void deletePreviaturas(Previaturas &lp, int id);		
	void PreviaturasIndexCrearGrafo(PreviaturasIndex &grafo);		
	bool PreviaturasIndexIsVerticeGrafo(PreviaturasIndex grafo, int id);		
	bool PreviaturasIndexIsAristaGrafo(PreviaturasIndex grafo, int id1, int id2);		
	void PreviaturasIndexInsertVerticeGrafo(PreviaturasIndex &grafo, int id);		
	void PreviaturasIndexInsertAristaGrafo(PreviaturasIndex &grafo, int id1, int id2);		
	int PreviaturasIndexGradoVerticeGrafo(PreviaturasIndex grafo, int id);		
	void deleteAllPreviaturasIndex(PreviaturasIndex &grafo);		
	void puedeInsertarPreviatura(PreviaturasIndex grafo, int a1, int a2, int &isErrorPorPertenenciaDirecta, bool &puedo);		
	void getAllPreviaturasByID(PreviaturasIndex grafo, int a1, Previaturas &pvs, bool visitado[N]);		
	bool tieneAlgunaPreviatura(PreviaturasIndex grafo, int a1);		
	void listAllPreviaturasByID(Previaturas pvs, Asignaturas asgs);		
	void crearPreviaturas(Previaturas &pvs);		
	void makePreviaturasIndex(PreviaturasIndex &grafo);		
	void ordenarPreviaturas(Previaturas &pvs);		


UDE
UNIVERSIDAD DE LA EMPRESA