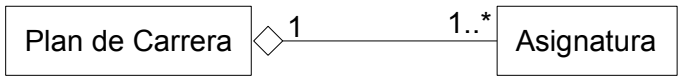
Segundo Certamen: Inicio: 11:45 hrs, término: 13:25 hrs.
Todas las preguntas tienen igual puntaje. No se atienden preguntas. Si estima que algo no se entiende o está ambiguo, haga una suposición razonable, escríbala y continúe.

1.- Responda en forma precisa y clara:
a) Señale dos ventajas de la metodología de desarrollo iterativo e incremental frente a un desarrollo en cascada.
La metodología de desarrollo iterativo e incrementar permite:

- *i) Mostrar resultados parciales al cliente y con ello validar el software, esto es que el software en desarrollo está satisfaciendo las necesidades del cliente. El desarrollo en cascada el cliente conoce la versión terminada basada en los requerimientos iniciales.*
- *ii) Permite hacer una mejor gestión del proyecto en cuanto a identificar problemas y soluciones a tiempo, pues en cada iteración se generan prototipos ejecutables en lugar tener sólo uno al final como el desarrollo en cascada y sólo allí detectar problemas.*

b) ¿Qué tipo de relación entre clases representa mejor la relación entre la clase "Asignatura" y la clase "Plan_de_Carrera"? Justifique. Nota: un plan de carrera es el conjunto de todas las asignaturas de una carrera.
*La relación que mejor representa esta relación es la **agregación**. En notación UML:*



No del todo correcto es composición. Es mejor agregación pues una asignatura tiene sentido por si sola y puede formar parte de otro plan de carrera. Si no fuera así, es decir la parte sólo tiene sentido dentro del todo, entonces sería composición.

c) Se dispone de los siguientes archivos: trabajo.cpp; trabajo.h, y el programa de prueba testTrabajo.cpp. Indique:

(i) cuál sería el comando para compilar testTrabajo.ccp.
Se pide compilar, luego sería en lo básico:
$ g++ -c testTrabajo.cpp
El resultado sería la aparición de testTrabajo.o en el directorio.

(ii) Cuál(es) sería(n) el (los) comando(s) para generar el ejecutable de nombre testTrabajo.
Para generar el ejecutable se debe compilar y ligar. Todo se puede hacer en un solo comando pero debemos poner como entrada todos los programas del cual el ejecutable depende:
$ g++ -o testTrabajo testTrabajo.cpp trabajo.cpp
Con -o definimos el nombre del ejecutable, de otro modo sería a.out. Debemos poner el archivo trabajo.cpp, pues se debe entender que testTrabajo.cpp al ser un programa de prueba de trabajo invocará cosas de éste.

d) En C alguien tiene la función void foo(EstructuraEstudiante student){ ...} donde student es un parámetro de entrada. Indique cuál es una declaración eficiente en C++ para esta función. Justifique.
*void foo(**const** EstructuraEstudiante **&** student) {...}*

e) Mencione dos ventajas de usar Patrones de Diseño.

- *(i) Ayuda a generar mejores soluciones al reutilizar soluciones bien pensadas para problemas recurrentes. Es la idea de reutilizar diseños, no sólo código.*
- *(ii) Ayuda a documentar el desarrollo pues los patrones de diseño son conocidos y están bien documentados.*
- *(iii) Proveen un lenguaje común entre diseñadores.*

2.- Se tiene la siguiente definición e implementación parcial de la clase Nodo que es la base de listas de enteros.

class Nodo
{
public:
  Nodo(){a=0; vecino=NULL; }
	Códigos posibles de uso:
	
	Caso a)
	  Nodo n1(3);

```
  Nodo(int a){ this->a=a; vecino=NULL;}
  void incremente();
    :
private:
  int a;
  Nodo * vecino;
};
void Nodo::incremente() {
 a++;
 for (Nodo *pn=vecino; pn!=NULL; pn=pn->vecino)
   (*pn).a++;
}
```

```
        Nodo n2=n1;

Caso b)
        cout << "los elementos de la lista son:"<< n1;

caso c)
        {
          Nodo  n3;
          n3 = n1;  // n1 viene de antes
          n3.incremente();  // manipula sólo n3
          :  // otras operaciones
        }
```

Complete la definición e implementación de la clase Nodo para permitir los usos señalados en los casos a), b), y c) incluso si nuevos métodos son agregados a la clase a futuro y los atributos privados se mantienen.

*Caso a) requiere definición e implementación del **constructor copia**.*
*Caso b) requiere sobrecarga del **operador <<**.*
*Caso c) requiere sobre-montar o redefinir el **operador asignación** =. Se debe hacer una copia completa para que la siguiente instrucción no afecte a n1, dado que hay atributo punteros.*
*Otra consecuencia del caso 3 es la implementación del **destructor**, pues sólo así los nodos creados en n3 durante la asignación liberan su memoria.*

```
class Nodo
{
public:
  Nodo(){a=0; vecino=NULL; }
  Nodo(Nodo&);                                          // caso a) requiere construcción copia. 3tps
  Nodo(int a){ this->a=a;vecino=NULL;}
 ~Nodo();                                               //  caso c) requiere definición del destructor 3pts.

  friend ostream& operator << (ostream &os, const Nodo &n);     // caso b) requiere sobrecarga del operador << , 3pts
  const Nodo& operator = (const Nodo&n);                // caso c) requiere definición del operador asignación 3pts
  void incremente();
private:
  int a;
  Nodo * vecino;
};

void Nodo::incremente()
{
 a++;
 for (Nodo *pn=vecino; pn!=NULL; pn=pn->vecino)
   (*pn).a++;
}
// constructor copia   + 3 pts.
Nodo::Nodo(Nodo&n)
{ Nodo * yo = this;
  a=n.a;
  vecino=NULL;
  for (Nodo *pn=n.vecino; pn!=NULL; pn=pn->vecino){
    yo->vecino= new Nodo(pn->a);
    yo=yo->vecino;
  }
}
// Destructor  +4 pts
Nodo::~Nodo()
{
  Nodo * pn=vecino, *paux;
  while(pn!=NULL) {
```

```
    paux=pn;
    pn=pn->vecino;
    delete paux;
  }
}
// Operador asignación + 3 pts.
const Nodo& Nodo::operator=(const Nodo &n)
{
  Nodo * yo=this;
  a=n.a;
  Nodo * pn=vecino, *paux;                          // en la primera parte se debe liberar el espacio actual.
  while(pn!=NULL) {
    paux=pn;
    pn=pn->vecino;
    delete paux;
  }                                                 // luego se deben copiar los nodos.
  for (Nodo *pn=n.vecino; pn!=NULL; pn=pn->vecino){
    yo->vecino= new Nodo(pn->a);
    yo=yo->vecino;
  }
  return *this;                                     // es importante retornar el resultado para admitir a=b=c;
}
// Operador <<   3pts
ostream & operator <<(ostream & os, const Nodo & n)  // pudo ser implementado de otra forma.
{                                                   // sólo debe cumplir con enviar a ostream los datos de la lista.
  os << "(" << n.a;
  for (Nodo *pn=n.vecino; pn!=NULL; pn=pn->vecino)
    os << " ," <<pn->a;
  os << ")" << endl;
  return os;
}
```

3.- Es común tener arreglos donde se desea determinar el número veces que un dato está presente. Considere la función cuentaRepeticiones. Si tenemos un arreglo A de números enteros ésta se podría usar en:
int total; cuentaRepeticiones(A, 3, total);
y en total obtenemos las veces que 3 está en el arreglo A.
a) Cree una template para poner invocar cuentaRepeticiones con otros tipos de datos para el mismo propósito.
b) Liste los requerimientos, si los hay, a cumplir por los tipos de datos que deseen usar su template o plantilla.
Nota: Suponga que no tiene permitido usar algoritmos genéricos.

a) 15 pts. Aquí hay un detalle que requiere una suposición. Uno de los argumentos es un arreglo y en C++, como en C, los arreglos no incluyen su tamaño; luego éste debe ser ingresado a la función. En la solución mostrada aquí se supone que tal tamaño es ingresado en el argumento total. Otras formas de abordar esto también son aceptables.

```
template <class T>
void cuentaRepeticiones(const T A[], const T & buscado, int & total) // total debe
                                                                     // ser referencia
                                                                     // para retornar el valor
{
  int maxIndice = total;
  total=0;
  for (int i=0; i<maxIndice; i++)
    if (A[i]==buscado) total++;
}
```

b) 10 pts. Para que esta template pueda ser usada, el tipo de dato del arreglo debe implementar el operador de comparación ==.

Explicación adicional: Ejemplos de uso de esta implementación se muestra abajo. Esto obviamente no se pide en la pregunta.

```
#include <iostream>
using namespace std;

template <class T>
void cuentaRepeticiones(const T A[], const T & buscado, int & total)
{
  int maxIndice = total;
  total=0;
  for (int i=0; i<maxIndice; i++)
    if (A[i]==buscado) total++;
}

class Estudiante
{
public:
  Estudiante()  {   nombre="";   } // por simplicidad código inline
  Estudiante (string nom)  {  // código inlin, pudo ser de otra forma
      nombre=nom;
  }
  bool operator==(const Estudiante & e)  const  {  // inline, pero pudo no serlo
      return e.nombre==nombre;
  }
private:
  string nombre;
};

int main(void)
{
  int A[20];
  int total;

  A[1]=A[5]=A[7]=3;
  total=20;
  cuentaRepeticiones(A, 3, total);  // prueba de invocación similar a pregunta
  cout << total<<endl;

  Estudiante B[30];
  Estudiante e("Eduardo");

  B[1]=B[5]=B[7]=B[14]=e;
  total=30;
  cuentaRepeticiones(B, e, total);  // invocación con arreglo de objetos
  cout << total<<endl;
}
```

4.- Usando el algoritmo genérico find_if(iterator start, iterator stop, unary function), el cual retorna un iterador al primer valor para la cual la función unaria es verdadera, implemente la función Buscar que en un vector de estudiantes encuentra aquel con un RUT dado. Implemente las funciones o clases que requiera.

class Estudiante {
public:
  : // otro métodos
  string getRUT();
  : // otras declaraciones y/0 métodos
}
	Se quiere poder invocar:
	vector<Estudiante> ve;
	:
	Estudiante *e=Buscar(ve, “12345678”);

Notar que por la invocación no se pide incluir un método sino crear una función. Esta función debe retornar un puntero a Estudiante y recibe un vector de estudiantes y string como argumentos.

```
class myClass
{
public:
  myClass (string r){
    rut=r;
  }
  bool operator() (Estudiante & e) {
    return rut==e.getRUT();
  }
private:
  string rut;
};

Estudiante* Buscar (vector<Estudiante> &ve, string rut){
  myClass myFunction(rut);
  vector<Estudiante>::iterator buscado;
  buscado = find_if(ve.begin(), ve.end(), myFunction);
  if (buscado==ve.end())
     return NULL;  // no está
  else
     return &(*buscado);  // *buscador accede al Estudiante encontrado.
          // Si alguien retorna
}         //  find..();  es considerado bien.
```

Nota: ver ejemplo presentado en clase del 8/07/08