Segundo Certamen

Primera parte, sin apuntes (30 minutos; 1/3 de la nota):

1.- Responda brevemente y entregue en hoja con su nombre.

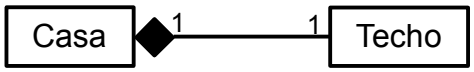
a. En la tarea 3 se eliminó la opción de la tarea 2 que permitía guardar los datos en un archivo. ¿Se podría haber incluido junto a los otros requerimientos de la tarea 3? Justifique su respuesta.
No, un applet no puede guardar datos en disco excepto que se cambien las políticas de seguridad.
Sí, existen mecanismos en Java para solicitar autorización al cliente y escribir datos en disco.

b. Un desarrollador prueba un applet subiéndola a su servidor web y luego accede a la página que la contiene. Al hacer sucesivas correcciones él vuelve a subir el archivo y observa que su applet pareciera no cambiar en el navegador. ¿Qué explica este comportamiento? ¿Cómo se soluciona?
¿Qué explica este comportamiento? Lo explica el cache que existe para las applets que corre un navegador. Una vez cargada un applet, futuras referencias a ella son tomadas desde el cache.
¿Cómo se soluciona? A través de la consola de Java del navegador se debe ejecutar el comando x para limpiar el cache.

c. ¿Cuáles ~~con~~ son los requerimientos funcionales? Dé un ejemplo de requerimiento **no** funcional.
Los requerimientos funcionales describen el comportamiento de un sistema ante determinadas entradas.
Un ejemplo de comportamiento no funcional es el lenguaje a usar (también el desempeño del sistema, -tiempo de ejecución, tamaño de código)

d. ¿Quiénes ~~nos~~ son los actores en un caso de uso?
Los actores son los entes externos que interactúan con el sistema.

e. A través de un esquema UML exprese la relación entre la clase Casa y la clase Techo.



Existe Composición. Es más específico que agregación porque un techo sólo forma parte de una casa. Cada instancia de Casa tiene un techo y éste techo le es propio. Si la casa no existe, esa instancia de techo tampoco existe.

f. Explique por qué al sobrecargar el operador << , para permitir salida de objetos hacia cout, estamos obligados a hacerlo como función global.
Pues sólo así podemos usar la sintaxis habitual para salida de datos, como en:
cout << objeto;
Si la sobrecarga la hiciéramos como método dentro de la clase del objeto, la sintaxis sería:
objeto << cout; // no respeta interpretación habitual de operador.
La opción de incluir el método dentro de la clase ostream se descarta por tratarse de una clase estándar del lenguaje.

g. ¿Qué diferencias existen entre ocupar una estructura (struct Student) o una clase (class Student) en un programa C++?
Sólo una, con clases por omisión los atributos y métodos son privados, con estructuras por omisión éstos son públicos.

h. ¿Bajo qué condiciones usted está obligado a implementar el destructor? ¿Qué otros métodos deben ser implementados adicionalmente?
Cuando la clase posee atributos alojados en espacios de memoria asignados dinámicamente que deben ser retornados al sistema.
Otros métodos a implementar en estos casos son: operador de asignación y constructor copia.

i. ¿Qué diferencia hace el definir un método como **virtual**?
Un método virtual permite que su redefinición en clases derivadas sean ligadas dinámicamente. Su ausencia evita el ligado dinámico, con lo cual en tiempo de ejecución se ejecuta el método según la declaración de la referencia y no según el objeto referenciado.

Segunda Parte, con apuntes (60 minutos, 2/3 de la nota. 1/3 + 1/3):

En preguntas 2 y 3 el puntaje está asignado sobre 100

2. Implemente la clase Complejo. La idea es ~~es~~ permitir las siguientes operaciones:
Complejo a(1,2),b(3,4),c; // **a** tiene parte real igual a 1 e imaginaria igual a 2.
c=a+b; // suma de complejos
c= a* // asigna el complejo conjugado. El conjugado sólo cambia el signo de la parte imaginaria.
c=a*b; // producto de complejos (recordar que i*i = -1)
cout << b; // que salga parte_real + parte_imaginaria i, como 3+4i

Nota: cometí un error al asumir que el operador * podía utilizarse post-fijo. Si esto fuera posible, la respuesta sería:

```
class Complejo {
public:    // 5 pts
  Complejo () { // constructor básico, para permitir Complejo c;
      x=y=0.0;
  };
  Complejo (float real, float imag);  // si usó double o int es OK aquí 10 pts.
  Complejo operator+ (const Complejo &c) const;   // 10 pts
  Complejo operator*(int)const;  // (OJO * no puede ser port-fijo) No será evaluado
  Complejo operator*(const Complejo &c )const;   // 10 pts
  friend ostream & operator<< (ostream &os, const Complejo& c); // 10 pts
private:  // 5 pts
  float x,y;    // 5 pts
};
Complejo::Complejo (float real, float imag) {  // 10 pts
  x = real;
  y = imag;
}
Complejo Complejo::operator +(const Complejo &c) const {  // 10 pts
  Complejo tmp(x+c.x, y+c.y);
  return tmp;
}
Complejo Complejo::operator*(int i) const {  // es un error * ,no puede ser post-fijo
  Complejo c(x,-y);                          // No será evaluado
  return c;
}
Complejo Complejo::operator *(const Complejo &c) const {  // 10 pts
  Complejo tmp(x*c.x-y*c.y, x*c.y+y*c.x);
  return tmp;
}
ostream & operator<< (ostream &os, const Complejo & c){  //15 pts
  os << c.x << "+" ;
  os << c.y << "i";
  return (os);
}
```

Nota: sí es sido válido usar *a para el caso de complejo conjugado, en ese caso la sobrecarga sería:

```
  Complejo operator*()const;
```

en la definición de la clase, y en la implementación:

```
Complejo Complejo::operator*() const {
  Complejo c(x,-y);
  return c;
}
```

3. Se desea implementar la función genérica maximo(vec) la cual recibe un vector ***vec*** (no vacío) de objetos y retorna una referencia constante (no un puntero) al mayor de ellos según el criterio de comparación que corresponda al objeto.

a) Cree una plantilla (template) para la función global maximo(vec) de manera que se pueda invocar con vectores de objetos, todos instancias de la misma clase.
b) Complete e implemente la clase Dado para que podamos hacer:
const Dado &maxDado = maximo(v); // cuando v es un vector de Dados.

```
class Dado{
public:
  Dado();  // por omisión el dado tiene 6 caras
  Dado(int n_caras); // crea dado de  n_caras
 :   // otros métodos que puedan ser requeridos para el uso pedido.
private:
     cont int numCaras;
     int valor;
} ;
```

a) 50 pts.

```
template <class T>   // 10 pts.
const T& maximo(const vector<T> &v) // es importante: tipos de datos 20 pts
{  int max = 0;                     // implementaciones 20 pts.
  for (int i=1; i<v.size(); i++)
    if (v[i]>v[max])
      max=i;
  return v[max];
}
```

b) 50 pts. Implementación de Dado para la implementación de la plantilla en a). Otras implementaciones podrían requerir métodos adicionales.

```
class Dado {   // Definición coherente con plantilla 20 pts
public:
  Dado();
  Dado(int n_caras);
  bool operator> (const Dado & d) const;  // Requerido por la platilla.
    // otros métodos
  const Dado & operator=(const Dado&d);  // no se pedía. Es requerida por vector.
                                         // La asignación por omisión no sirve por
                                         // tener atributo constante.
private:
  const int numCaras;
  int valor;
};

Dado::Dado()
    :numCaras(6)  {    // 8 pts
  valor=random()%6+1;  // no importa para el problema
}
Dado::Dado(int n_caras)  // 7 pts  Ambas son parecidas
    :numCaras(n_caras) {
  valor=random()%n_caras+1;   // no importa para el problema
}
bool Dado::operator>(const Dado & d) const {  // 15 pts
  return valor > d.valor;  // no importa su criterio de comparación
}
const Dado & Dado::operator=(const Dado&d){  // no se pedía
  valor=d.valor>numCaras?numCaras:d.valor;
  return *this;
}
```