Certamen Final
Todas las preguntas tiene el mismo puntaje. Tiempo 90 Minutos.

1.- Se tiene la siguiente clase para nombre:

```
class Nombre {
public: Nombre () {pila=""; apellido="";};
public:
        string pila;
        string apellido;
}
```

Haga un programa en C++ que lea desde el teclado nombre y apellido de personas y los almacene en una lista. Luego se pide listar los nombres completos ordenamos por apellido y para igual apellido ordenados según nombre de pila.
La clase Nombre puede ser extendida según se necesite.

```
#include <list>
#include <string>

class Nombre {
public:
  Nombre () {pila=""; apellido="";}
  bool operator < (Nombre &n)
  {
    if (apellido < n.apellido)
      return true;
    else if (apellido == n.apellido)
      return pila < n.pila;
  }
public:
  string pila;
  string apellido;
};

void main(void)
{
  list <Nombre> nameList;

  cout << "por favor ingrese nombre y apellido. Termine con un `.` en la línea "<<endl;

  while(true) {
    Nombre* name=new Nombre();

    cin >> name->pila;
    if (name->pila==".") break;
    cin >> name->apellido;
    nameList.push_back(*name);
  }

  nameList.sort();
  cout << "Listado de nombres ordenados por apellido\n";

  for (list<Nombre>::iterator itr=nameList.begin(); itr!=nameList.end();itr++)
    cout << itr->pila <<" " << itr->apellido << endl;
}
```

2.- a) ¿Cuál es la diferencia entre las llamadas v.capacity() y v.size(), cuando v es un vector?

v.capacity() retorna en número de entradas que el vector puede alojar en el espacio de memoria que ya ha sido asignado para él. Es siempre mayor o igual que v.size(). v.size() retorna el número de entradas que están siendo efectivamente usadas en el vector v.

b) Si tenemos: **vector<int> items ;**

Muestre el segmento de código que permite eliminar el menor de los elementos del vector usando algoritmos genéricos.

```
vector<int>::iterator iter =min_element(items.begin(), items.end());
tiems.erase(iter);
```

c) ¿Para qué se usa y qué significado tiene el calificador **final** en Java?

- **Se usa final cuando deseamos evitar el ligado dinámico.**
- **Cuando deseamos que el valor de una variable escalar permanezca sin cambio.**
- **Cuando deseamos que un nombre asociado a un objeto no cambie.**
- **Lo usamos con clases cuando queremos que ésta no dé origen a clases derivadas.**

d) Si tenemos en Java una clase Punto y hacemos: Punto p;
¿Cuántos elementos característicos de todo objeto se han definido? Nota: Los elementos que caracterizan a todo objeto son: nombre, atributos y comportamiento.

El nombre del objeto. También se podría decir que está definido el comportamiento; es decir los métodos que podemos invocar sobre p; sin embargo, la implementación de estos métodos no necesariamente serán los de la clase Punto, ya que otras clases podrían derivar de punto e instancias con distinta implementación podrían ser asignadas a punto.

3.- Se tienen las siguientes funciones que caracterizan el comportamiento de lagunas componentes digitales.
Asuma las siguinets caracterizaciones:
Reseteable: este comportamiento define que un componente se pude "resetear", dispone del método **reset**().
Desplazable: este comportamiento define que los datos de un componente se pueden desplazar a la derecha e izquierda, dispone de los métodos **boolean shiftLeft(boolean in)** y **boolean shiftRight(boolean in)**. En ambos casos se ingresa el bit pasado como parámetro y se retorna el bit de más a la izquierda o más a la derecha que sale del componente.

a) Implemente interfaces que caractericen los comportamientos indicados (Reseteable, Desplazable).
b) Implemente la clase RegistroDesplazamiento. La cual ofrece los comportamientos de estas dos interfaces. Usted es libre de elegir como almacenará los bits del registro universal.

a)En archivo Reseteable.java:

```
public interface Reseteable
{
  public void reset();
}
```

En archivo Desplazable.java:

```
public interface Desplazable
```

```
{
  public boolean shiftRight(boolean in);
  public boolean  shiftLeft(boolean in);
}

b)public class p3 implements Reseteable, Desplazable
{
  private boolean bits[];
  public p3 () {
    bits=new boolean[8];  // bits[0],bits[1],..,bits[7]
  }
  public  void reset() {
    for (int i=0; i<8; i++) bits[i]=false;
  }
  public boolean shiftLeft(boolean in)
  {
    boolean ret=bits[0];
    for (int i=0; i<7; i++)
      bits[i]=bits[i+1];
    bits[7]=in;
    return ret;
  }

  public boolean shiftRight(boolean in)
  {
    boolean ret = bits[7];
    for (int i=7; i>0; i--)
      bits[i]=bits[i-1];
    bits[0]=in;
    return ret;
  }
  public String toString(){ // Sólo para probar implementación
    String sal="";
    for (int i=0; i<8; i++)
      sal+= bits[i]+", ";
    return sal;
  }

  public static void main (String arg[])  // Sólo para probar implementación
  {
    p3 registro = new p3();
    for (int i=0; i<8; i++)
      registro.shiftRight(i%2==0);
    System.out.println(registro);
  }
}
```

4) Desarrolle un applet que muestre un contador que se incremente cada 100 ms partiendo de 0. Se debe disponer también de un botón que permita pausar el contador y con el mismo botón se pueda resumir su cuenta.
Escriba una página html que permita su visualización.

```
///////    p4.java    ///////
import java.applet.Applet;
import java.awt.event.*;
import java.awt.*;

public class p4 extends Applet
{
```

```
  public TextField text;
  public Button stop_resume;

  public void init()
  {
    stop_resume = new Button("  stop  ");
    text = new TextField(5);
    stop_resume.addActionListener(new MyActionListener(this));
    add(stop_resume);
    add(text);
  }
}

class MyActionListener extends Thread implements ActionListener
{
  int count;
  p4 applet;
  boolean stop;

  public MyActionListener(p4 app)
  {
    count=0;
    applet = app;
    stop=false;
    start();
  }

  public void run()
  {
    try {
      while(true){
            if(!stop) {
              applet.text.setText(" "+ (count++));
              applet.repaint();
            }
            sleep(100);
      }
    } catch(InterruptedException e) {}
  }

  public void actionPerformed(ActionEvent e)
  {
    stop= !stop;
    if (stop)
      applet.stop_resume.setLabel("resume");
    else {
      applet.stop_resume.setLabel("stop");
    }
    applet.repaint();
  }
}
```

Archivo HTML:

```
<html>
Si usted presiona el botón, el contador se detendrá o resumirá.<br>
<applet Code="p4.class" width=300 height=80 > </applet>
</html>
```

Alternativamente se pudo resolver más eficientemente así:

```
///////   p4v2.java   ///////
import java.applet.Applet;
import java.awt.event.*;
import java.awt.*;

public class p4v2 extends Applet
{
  public TextField text;
  public Button stop_resume;

  public void init()
  {
    stop_resume = new Button("  stop  ");
    text = new TextField(5);
    stop_resume.addActionListener(new MyActionListener(this));
    add(stop_resume);
    add(text);
  }
}

class MyActionListener extends Thread implements ActionListener
{
  int count;
  p4v2 applet;
  boolean stop;

  public MyActionListener(p4v2 app)
  {
    count=0;
    applet = app;
    stop=false;
    start();
  }

  public void run()
  {
    try {
      while(true){
            while(!stop) {
              applet.text.setText(" "+ (count++));
              applet.repaint();
              sleep(100);
            }
            synchronized (this) {
              wait();
            }
      }
    } catch(InterruptedException e) {}
  }

  synchronized public void actionPerformed(ActionEvent e)
  {
    stop= !stop;
    if (stop)
      applet.stop_resume.setLabel("resume");
    else {
      applet.stop_resume.setLabel("stop");
      notify();
    }
    applet.repaint();
```

```
  }
}
```