Prob.1. En la práctica 4 (¿Qué impresora uso?), se nos acaba de romper la impresora nueva. Ante esta situación nos piden que realicemos una nueva función para el programa de gestión de colas de impresión que pase los trabajos que quedan en la cola de impresión de la impresora nueva a la cola de impresión de la impresora vieja (hasta un máximo de 10, ya que no cabían más en la cola de la impresora vieja). El resto se perderán. Estos ficheros que se pierden y no pueden ser impresos se mostrarán por pantalla. El prototipo de la función será:

```
void RecogerTrabajos (Cola &, Cola &);
```

```
/*************** RecogerTrabajos*****************************************
 *
 *  Descripcion: Esta funcion pasa los trabajos de la cola de impresion
 *               de la impresora nueva a la cola de impresión de la
 *               impresora vieja (hasta su maxima capacidad de 10
 *               trabajos). Los trabajos que no podamos pasar los
 *               mostraremos por pantalla.
 *
 *  Parametros:
 *  Nombre          Tipo        E/S  Descripcion
 *  ------          ----------  ---  ------------
 *  impre_nueva     Cola        E S  Cola que contiene los trabajos de
 *                                   impresión de la impresora nueva
 *  impre_vieja     Cola        E S  Cola que contiene los trabajos de
 *                                   impresión de la impresora vieja
 *
 *  Valor de retorno:
 *  ninguno
 *
 ************************************************************************/
void RecogerTrabajos (Cola & impre_nueva, Cola & impre_vieja)
{
    string nomfich;

    // Mientras la cola de impresión de la impresora nueva tenga trabajos
    while (!impre_nueva.ColaVacia () )
    {
        // Consultamos su primer elemento
        impre_nueva.PrimeroCola (nomfich);

        // Si la cola de impresión de la impresora vieja ya esta llena
        // no podremos encolar el trabajo e informamos al usuario
        if (!impre_vieja.Encolar (nomfich) )
            cout << "Se va a perder el fichero " << nomfich << endl;

        // Una vez gestionado el trabajo lo eliminamos de la cola de impresion
        impre_nueva.Desencolar ();
    }
    return;
}
```

Prob.2.

a. Explica por qué en el algoritmo para determinar los puntos de ruptura de un grafo, debemos señalar como visitado el nodo que queremos comprobar si es o no de ruptura antes de pasar el algoritmo BFS.

Debemos señalarlo, para que cuando intentemos atravesarlo con el algoritmo BFS, al estar el nodo ya visitado no podamos atravesarlo, y consideremos a todos los efectos como si lo huviesemos borrado del nodo.

b. ¿Se podrían determinar los puntos de ruptura utilizando el algoritmo DFS? ¿Cambiaría algo del algoritmo si lo hiciésemos? Razona las respuestas.

Si, y no cambiaría en absoluto el algoritmo de determinación de puntos de ruptura, ya que el algoritmo de recorrido sólo nos sirve para determinar si hemos podido o no recorrer por completo el grafo. Si hemos conseguido recorrer por completo el grafo (da igual si con el algoritmo BFS o con el algoritmo DFS) el punto no era de ruptura, mientras que si no lo hemos conseguido el punto si que era de ruptura.

Prob.3. Teniendo en cuenta la siguiente definición de la clase CARTA:

```
class Carta
{
   public:
      Carta ();

      bool ponerCarta (char, int);

      //devuelve el palo de la carta
      char paloCarta ();
      //devuelve el numero de la carta
      int numeroCarta ();
      //muestra por pantalla la carta
      void MostrarCarta ();

   private:
      // 'O'=oros, 'C'=copas,
      // 'E'=espadas, 'B'=bastos
      char palo;
      //1..12
      int numero;
};
```

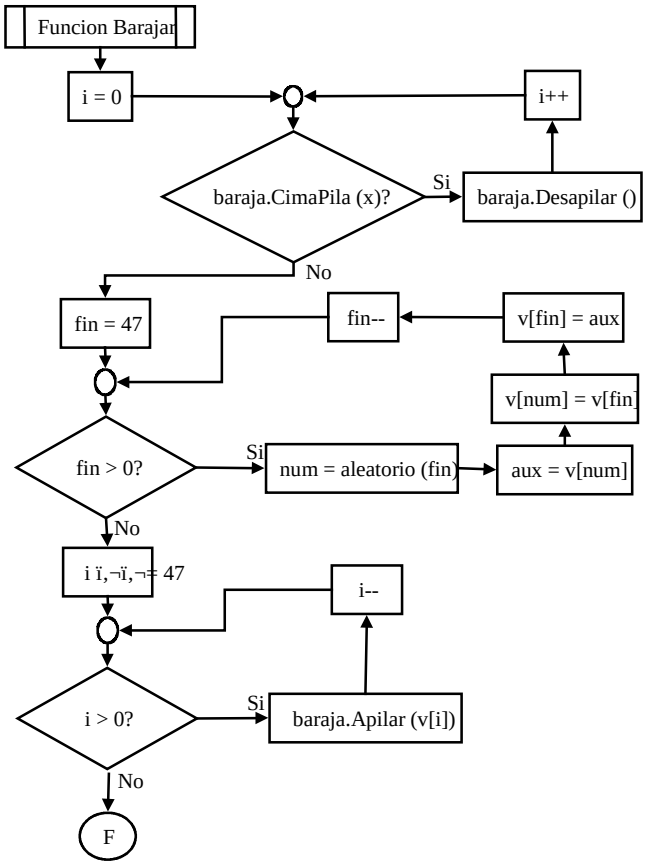
y la definición de la clase PILA:

```
class Pila
{
   public:
      Pila ();

      bool Apilar (Carta);
      bool Desapilar ();
      bool CimaPila (Carta &);
      bool PilaVacia ();

      void MostrarPila ();
      int NElementosPila ();

   private:

      typedef Carta Vector[MAX];
      Vector datos;
      int cima;
   };
```

a. Realiza el diagrama de flujo de la función **`Barajar`** que, utilizando el siguiente algoritmo, debe barajar la pila de cartas que se pasa como parámetro:

1.- Volcar los datos de la pila en un vector auxiliar v de cartas (la pila queda vacía)
2.- Desde fin = 47 hasta 1 hacer
    2.1.- pos = generar un nº aleatorio entre 0 y fin
    2.2.- intercambiar los elementos v[pos] y v[fin]
3.- Volcar los datos de v de nuevo en la pila

Para realizar el algoritmo supondremos la existencia de la función '**`int aleatorio (int max);`**' que determina números aleatorios entre '**`0`**' y '**`max – 1`**'.



b. Implementa la función **Barajar** en C++. El prototipo de la función será:

```
void Barajar (Pila &);

typedef Carta Vect_Cartas[48];

/*************** Barajar ********************************************
 *
 *  Descripcion: A partir de la información contenida en la pila (baraja
 *               ordenada) se baraja y se obtiene una baraja desordenada
 *
 *  Parametros:
 *  Nombre          Tipo        E/S  Descripcion
 *  ------          ----------  ---  ------------
 *  baraja          Pila        E S  Pila que contiene la baraja ordenada
 *                                   que vamos a barajar
 *
 *  Valor de retorno:
 *  ninguno
 *
 ********************************************************************/
void Barajar (Pila & baraja)
{
    int num;            // Entero donde guardaremos numeros aleatorios generados
    Carta aux;          // Carta auxiliar para realizar intercambios
    Vect_Cartas v[48];  // Vector auxiliar de cartas
    int fin, i;         // Variables auxiliares para los bucles

    // Volcamos los datos de la pila en el vector auxiliar v
    i = 0;
    while (baraja.CimaPila(v[i]) )
    {
        baraja.Desapilar();
        i++;
    }

    for (fin = 47; fin > 0; fin--)
    {
        num = aleatorio (fin);
        aux = v[num];
        v[num]= v[fin];
        v[fin] = aux;
    }

    // Devolvemos los elementos que tenemos en el vector a la baraja
    for (i = 47; i > 0; i--)
        baraja.Apilar(v[i]);
}
```

Prob.4. En la práctica 5 (Generación de exámenes tipo test) el formato del fichero **`preguntas.txt`** era el siguiente

- **Pregunta (cadena de texto con la pregunta).**
- **Respuesta A (texto de la respuesta A).**
- **Respuesta B (texto de la respuesta B).**
- **Respuesta C (texto de la respuesta C).**
- **Respuesta correcta (LETRA A, B o C).**
- **Nivel de dificultad de la pregunta (valor entero, de menor a mayor dificultad: 1-5).**
- **Número de veces usada la pregunta en un test. (valor entero)**

Cada una de estas informaciones ocupa una línea en el archivo.

a. Define un tipo registro para almacenar esta información.

```
struct Valor
{
    string pregunta; // Enunciado de la pregunta
    string resA;     // Opcion de respuesta 'C'
    string resB;     // Opcion de respuesta 'C'
    string resC;     // Opcion de respuesta 'C'
    char correcta;   // Opcion correcta: 'A', 'B' o 'C'
    int nivel;       // Nivel de dificultad 1.-poca ... 5.-mucha
    int veces;       // Numero de veces que ha aparecido la pregunta en otros test
};
```

b. Implementa una función que reciba como argumento el nombre del fichero en un string y devuelva la información leída en una lista. El prototipo de la función será:

```
Lista LeerFichero (string);
```

```
/*************** LeerFichero ******************************************
 *
 *  Descripcion: Abre el fichero 'nombre' que guarda la informacion de
 *               las preguntas del test y las pone en una lista
 *
 *  Parametros:
 *  Nombre          Tipo        E/S  Descripcion
 *  ------          ----------  ---  ------------
 *  nombre          string      E    Nombre del fichero en el que estan
 *                                   las preguntas
 *
 *  Valor de retorno:
 *  Lista           Lista en la que esta la informacion leida del fichero
 *
 **********************************************************************/
Lista LeerFichero (string nombre)
{
    Valor x;
    Lista lis;
    ifstream f;

    f.open (nombre.c_str() );

    if (!f)
        cout << "Error al abrir el fichero\n";
    else
    {
        while (getline(f, x.pregunta) )
        {
            getline(f, x.resA);
            getline(f, x.resB);
            getline(f, x.resC);
            f >> x.correcta;
            f >> x.nivel;
            f >> x.veces;
            f.ignore();
            lis.Insertar(x);
        }
        f.close();
    }
    return (lis);
}
```