Tema 16: Obligatorio y defensa

Hasta este tema, existe una forma de enseñanza incremental, donde a cada tema le sigue uno de mayor complejidad, pero le falta algo indispensable para el curso de programación que es la necesidad de integrar los conocimientos adquiridos.
Para ello el docente debe proponer cerca de fin de año un trabajo Obligatorio.
Le llamamos Obligatorio a un programa que el estudiante debe realizar en equipo, en base a una propuesta, la cual debe entregar, documentar y defender a fin de año.
No le llamamos proyecto porque le faltan elementos para llegar a ello, en el diseño y avance del mismo. Dejamos el proyecto para años posteriores donde el estudiante está más maduro para estas tareas.
Equipo:
El estudiante debe formar un equipo de 1 a 3 estudiantes.
A pesar que un equipo de 1 está mal definido, en este primer año se aceptará esta opción.
Propuesta:
Para lograr resultados aceptables se sugiere proponer la letra del obligatorio después de vacaciones de primavera.
Esto requiere que antes de octubre esté casi terminado el programa, de manera de apoyar el mismo.
El apoyo consiste en dictar temas que permitan realizar eficientemente el obligatorio y atender en clase las consultas de los alumnos.

Entrega:
Se debe fijar una fecha de entrega del proyecto.
La misma generalmente es en la última clase de octubre.
Avance:
El alumno a medida que va avanzando en su obligatorio lo puede (y debe) ir llevando a clase para que el docente lo oriente.
Defensa:
Por lo general en noviembre se hace la defensa del obligatorio.
El docente hace pasar a la clase a los equipos, uno a uno, prueba el programa y les va preguntando a cada integrante ¿qué hace? ¿para qué sirve? ¿cómo solucionaría los errores del programa presentado? De manera de verificar el grado de participación en el mismo, el nivel de conocimientos de programación y si está preparado para cursar Programación 2.

Quien no presente el obligatorio debe ir a examen.

Reglas
Se acostumbra ajustar la entrega a ciertas reglas como ser:

1) No se aceptan variables en inglés. Esto está fundado en estudiantes que buscan soluciones en Internet que estan en ingles, la copian tal cual y la presentan.

2) Los alumnos deben presentar el código del programa impreso ya que eso es una documentación del trabajo. Debe figurar los nombres de los integrantes
3) Si se detecta copia en los trabajos, quienes copian y quienes se dejaron copiar van a examen.
4) Presentar el obligatorio y presentarse a defenderlo son condiciones para no ir a examen.

Ejemplo de Obligatorio

Passwords Aceptables

Un password seguro es algo delicado. Los usuarios prefieren passwords que sean fáciles de recordar (como amigo), pero este password puede ser inseguro. Algunos lugares usan un generador randómico de passwords (como xvtpzyo), pero los usuarios toman demasiado tiempo recordándolos y algunas veces lo escriben en una nota pegada en su computador. Una solución potencial es generar password “pronunciables” que sean relativamente seguros pero fáciles de recordar.
PassUTU está desarrollando un generador de passwords.
Su trabajo en el departamento de control de calidad es probar el generador y asegurarse de que los passwords sean aceptables. Para ser aceptable, el password debe satisfacer estas tres reglas:
1. Debe contener al menos una vocal.
2. No debe tener tres vocales consecutivas o tres consonantes consecutivas.
3. No debe tener dos ocurrencias consecutivas de la misma letra, excepto por ‘ee’ o ‘oo’.
(Para el propósito de este problema, las vocales son 'a', 'e','i', 'o', y 'u'; todas las demás letras son consonantes.)
Note que estas reglas no son perfectas; habrán muchas palabras comunes/pronunciables que no son aceptables.
La entrada consiste en una o más potenciales passwords, uno por línea, seguidas por una línea conteniendo una palabra 'fin' que señala el fin de la entrada. Cada password tiene como mínimo una y como máximo veinte letras de largo y esta formado solo por letras en minúscula. Por cada password, despliegue si es o no aceptable, usando el formato mostrado en el ejemplo de salida.

Ejemplo de entrada
a
tv
ptoui
bontres
zoggax
wiinq
eep
houctuh
fin

Ejemplo de salida
<a> es aceptado.
<tv> no fue aceptado.
<ptoui> no fue aceptado.
<bontres> no fue aceptado.
<zoggax> no fue aceptado.
<wiinq> no fue aceptado.
<eep> es aceptado.
<houctuh> es aceptado.

Solución

Este problema presenta tres simples reglas que se deben cumplir para que un password sea aceptable.
Primero es necesario identificar si un carácter es vocal o no, para lo cual utilizo la siguiente función que pregunta si el carácter es una vocal, y devuelve true por verdad y false por falso.

```
int esVocal( char ch ){
  if( ch=='a' || ch=='e' || ch=='i'|| ch=='o' || ch=='u' )
  return true;
return false;
}
```

Ahora iremos validando cada una de las tres condiciones expuestas en el planteamiento del problema en su respectivo orden.

Debe contener al menos una vocal.- Basta con realizar un recorrido por toda la cadena (palabra), en cuanto encontremos una vocal se retorna **true** y si se ha terminado de hacer
el recorrido y no se ha encontrado ninguna vocal retornamos **false**.

```
int regla_1(){
        int i;
        for( i=0; i<len; i++ )
                if( esVocal( palabra[i] ) )
                return true;
        return false;
}
```

No debe tener tres vocales consecutivas o tres consonantes consecutivas.-

Otra vez un recorrido por toda la cadena, pero esta vez utilizando dos contadores **v** y **c**, que se incrementan en cuanto se encuentra una vocal o consonante respectivamente, en cuanto alguno llegue a tres, la función termina con falso, y si logra terminar el recorrido sin problemas se da por cumplida la segunda regla.

```
int regla_2(){
        int i, v=0, c=0;
        for( i=0; i<len; i++ ){
                if( esVocal( palabra[i] ) ){
                  v++; c=0;}
                else{
                  c++; v=0;
                }
                if( v==3 || c==3 )
                  return false;
        }
return true;
}
```

No debe tener dos ocurrencias consecutivas de la misma letra, excepto por "ee" o "oo". Otro recorrido más, esta función, como las anteriores es muy explícita, la única diferencia es que empieza en el segundo elemento de la cadena y no así en el primero.

```
int regla_3(){
        int i;
        for( i=1; i<len; i++ ){
                if( palabra[i]==palabra[i-1] && palabra[i]!='e' && palabra[i]!='o' )
                return false;
        }
        return true;
}
```

Bien, ahora solo resta leer cadenas, verificamos si cumplen las tres reglas e imprimir el resultado correspondiente, en caso de haber leído la cadena 'fin', termina el programa.

```
import java.util.Scanner;
public class passw{
   static char palabra[];
   static int largo;

   static boolean esVocal( char letra ){
      if( letra=='a' || letra=='e' || letra=='i'
            || letra=='o' || letra=='u' )
         return true;
      return false;
   }

   static boolean regla_1(){
      for( int i=0; i<largo; i++ )
         if( esVocal( palabra[i] ) )
            return true;
      return false;
   }
```

```
    static boolean regla_2(){
        int i, v=0, c=0;
        for( i=0; i<largo; i++ ){
            if( esVocal( palabra[i] ) ){
                v++; c=0;
            }else{
                c++; v=0;
            }
            if( v==3 || c==3 )
                return false;
        }
        return true;
    }

    static boolean regla_3(){
        for( int i=1; i<largo; i++ ){
            if( palabra[i]==palabra[i-1]
                    && palabra[i]!='e' && palabra[i]!='o' )
                return false;
        }
        return true;
    }
    public static void main( String args[] ){
        String linea;
        Scanner in = new Scanner( System.in );
        while( true ){
            System.out.print("Ingrese un password ('fin'-termina) ");
            linea = in.next();
            if( linea.equals("fin") )
                break;
            palabra = linea.toCharArray();
            largo = linea.length();
            if(regla_1()&& regla_2() && regla_3())
                System.out.println( "<" + linea + "> es aceptado." );
            else
                System.out.println( "<" + linea + "> no fue aceptado.." );
        }
    }
}
```

¿Es posible hacerlo todo con una única función?

Claro, existen muchas formas de hacer las cosas.

```
int reglas_1_2_3(){
        int i, v=0, c=0, vocales=0;
        for( i=0; i<len; i++ ){
                if( palabra[i]=='a' || palabra[i]=='e'
                || palabra[i]=='i' || palabra[i]=='o'
                || palabra[i]=='u' ){
                v++; c=0; vocales++;
                }else{
                c++; v=0;
        }
        if( v==3 || c==3 )
                return 0;
        if( palabra[i]==palabra[i+1]&& palabra[i]!='e' && palabra[i]!='o' )
                return 0;
        }
        return vocales;
}
```

Esta función es más complicada y más difícil de entender. Se sugiere siempre hacer códigos legibles y depurables.