UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2020

Versión 1: 12 de junio de 2020

TPI - "Reuniones Remotas"

Entrega: 6 de julio (hasta las 17 hs)

1. Consignas

- Implementar todas las funciones que se encuentran en el archivo `ejercicios.h`. Para ello, deberán usar la especificación que se encuentra en la última sección del presente enunciado. La implementación de cada ejercicio DEBE SEGUIR OBLIGATORIAMENTE ESTA ESPECIFICACIÓN.
- Extender el conjunto de casos de tests de manera tal de lograr una cobertura de líneas mayor al 95 %. En caso de no poder alcanzarla, explicar el motivo. La cobertura debe estar chequeada con herramientas que se verán en laboratorio de la materia.
- No está permitido el uso de librerías de C++ fuera de las clásicas: **math**, **vector**, **tuple**, **pair**, las de input-output, etc. Consultar con la cátedra por cualquier librería adicional.

Dentro del archivo que se descarguen desde la página de la materia van a encontrar los siguientes archivos y carpetas:

- `definiciones.h`: Aquí están las definiciones de los tipos del TPI.
- `ejercicios.cpp`: Aquí es donde van a volcar sus implementaciones.
- `ejercicios.h`: *headers* de las funciones que tienen que implementar.
- `auxiliares.cpp` y `auxiliares.h`: Donde es posible volcar funciones auxiliares.
- `main.cpp`: Punto de entrada del programa.
- `tests`: Estos son algunos Tests Suites provistos por la materia. Aquí deben completar con sus propios Tests para lograr la cobertura pedida.
- `lib`: Todo lo necesario para correr Google Tests. Aquí no deben tocar nada.
- `CMakeLists.txt`: Archivo que necesita CLion para la compilación y ejecución del proyecto. **NO** deben sobreescribirlo al importar los fuentes desde CLion. Para ello recomendamos:
  1. Lanzar el CLION.
  2. Cerrar el proyecto si hubiese uno abierto por *default*: `File->Close Project`
  3. En la ventana de Bienvenida de CLION, seleccionar `Open Project`
  4. Seleccionar la carpeta del proyecto `template-alumnos`.
  5. Si es necesario, cargar el CMakeList.txt nuevamente mediante `Tools->CMake->Reload CMake Project`
  6. No olvidarse descomprimir el GTEST.

Es importante recalcar que la especificación de los ejercicios elaborada por la materia es la guía sobre la que debe basarse el equipo a la hora de implementar los problemas.

2. Entregable

La fecha de entrega del TPI es el **06 de Julio de 2020**.

1. Entregar una implementación de los ejercicios que cumplan el comportamiento detallado en la Especificación. Los archivos obligatorios que se deben entregan son: `ejercicios.cpp`, `auxiliares.cpp`, `auxiliares.h` y el `CMakeList.txt`. Además, incluir los casos de test adicionales propuestos por el grupo que debieron desarrollar para incrementar la cobertura.
2. El proyecto debe subirse en un archivo comprimido en la solapa Trabajos Prácticos en la tarea SUBIR TPI.
3. **Importante: Utilizar la especificación diseñada para este TP.**.
4. **Importante: Es condición necesaria que la implementación pase todos los casos de tests provistos en el directorio tests. Estos casos sirven de guía para la implementación, existiendo otros TESTS SUITES *secretos* en posesión de la materia que serán usados para la corrección.**

3. Introducción

En los tiempos que corren atravesamos la necesidad de conectarnos remotamente con otras personas, ya sea individidual o grupalmente. En este contexto usamos programas que nos permiten interactuar con nuestros interlocutores. Nos interesa resolver una serie de problemas relacionados con el habla, que se dan cuando un grupo de personas mantiene una conversación online. La voz de los interlocutores será captada a través de sus microfónos y almacenadas en secuencias de datos con las cuales vamos a trabajar.

4. Definiciones

Las ondas sonoras son variaciones continuas en la presión del aire, esas variaciones de presión se convierten a través de un micrófono en una señal eléctrica. Existen dos maneras de almacenar esas señales, de forma analógica o digital.

En este trabajo utilizaremos audios almacenados de manera digital representados con una secuencia de valores enteros a los que llamaremos muestra.

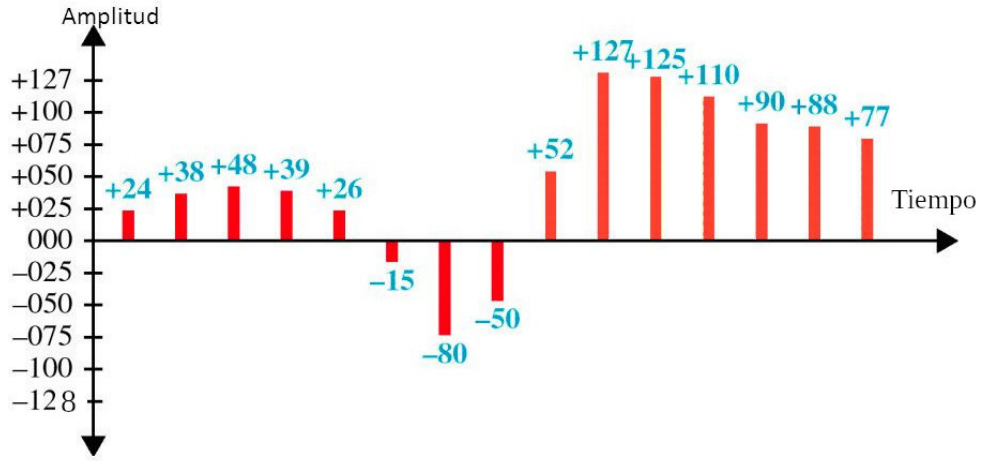
Cada muestra representa la amplitud de la señal en un punto específico en el tiempo. El rango de las muestras quedará determinado por el número de bits de información en cada una de ellas (profundidad). Siendo P la *profundidad*, el rango queda definido de la siguiente manera:

$$[-2^{(P-1)} \ , \ 2^{(P-1)} - 1 \ ]$$

Los Cd's utilizan P igual a 16, es decir, 16 bits por muestra, mientras que para el Blu-ray, P es 24.

Por último, la *frecuencia de muestreo* definirá la cantidad de veces que nuestro equipo recoge una muestra de la señal por segundo, 1000 muestras por segundo equivalen a 1 kHz. Algunos ejemplos: Teléfono (voz) 8 kHz, Radiodifusión 32 kHz, CD 44.1 kHz. A partir de ahora cuando hablemos de frecuencia estará medida en kHz.

En el siguiente ejemplo vemos una señal analógica codificada para muestras digitales de 8 bits; la profundidad de bits es ocho, por lo que la amplitud de cada muestra es uno de los 256 valores posibles dentro del rango $[-2^{(8-1)} \ , \ 2^{(8-1)} - 1 \ ]$ = [-128, 127]



Además contaremos con reuniones, una reunión es una secuencida de tuplas *señal* × *hablante* que representa la señal que tiene cada persona en una conversación. Dentro de una reunión, todas las conversaciones comienzan y terminan al mismo tiempo (tienen la misma cantidad de muestras). Además, no puede haber mas de una señal para la misma persona. Las personas estarán representadas con números, enteros empezando en 0 (en una reunión de n personas, se les asignarán los números en el rango $[0..n-1]$). Por ejemplo, si en una reunión participan 3 personas, las personas hablando se podrían representar con la secuencia [1, 0, 2].

5. Ejercicios

Dados el siguente renombre de tipos:

type *señal*= $seq\langle\mathbb{Z}\rangle$
type *hablante*= $\mathbb{Z}$
type *reunion*= $seq\langle señal \times hablante\rangle$

Especificar los siguentes problemas:

Ejercicio 1. **proc esSeñal(in s:** $seq\langle\mathbb{Z}\rangle$**, in prof :** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$**, out result :** Bool): dada una secuencia s, comprueba si es una señal. Diremos que una secuencia es una señal cuando la frecuencia de muestreo es 10 hz, la profundidad 8, 16 o 32 bits y dura más de 1 segundo.

Ejercicio 2. **proc seEnojó?(in s:** ***señal*****, in umbral:** $\mathbb{Z}$**, in prof :** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$**, out result:** Bool): dada la señal de un hablante indicará si este se enojó. Diremos que un hablante está enojado si a lo largo de la conversación existe un lapso de tiempo donde su tono supera el umbral, siendo el tono de voz el promedio de las amplitudes de la señal en valor absoluto. Este lapso debe ser de más de 2 segundos.

Ejercicio 3. **proc esReuniónVálida(in r:** ***reunion*****, in prof :** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$**, out result :** Bool): dada una reunión r, comprueba si es una reunión válida.

Ejercicio 4. **proc acelerar(inout r:** ***reunion*****, in prof:** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$): dada una reunión, acelera el hablar de cada uno de los participantes. Una señal será acelerada cuando posea la mitad de las muestras de la original, tomando solamente las muestras que se encuentran en posiciones impares.

Ejercicio 5. **proc ralentizar(inout r:** ***reunion*****, in prof:** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$): dada una reunión, lentificar el hablar de cada uno de los participantes. Una señal será lentificada cuando posea el doble de las muestras originales. Se crearan muestras ficticias entre medio de cada par de puntos de la señal, las mismas serán el promedio de los dos puntos vecinos.

Ejercicio 6. **proc tonosDeVozElevados(in r:** ***reunion*****, in freq:** $\mathbb{Z}$**, in prof :** $\mathbb{Z}$**, out hablantes:** $seq\langle$***hablante***$\rangle$): Dada una reunión, determina cuales fueron los hablantes que tienen el tono de voz más elevado.

Ejercicio 7. **proc ordenar(inout r:** ***reunion*****, in freq:** $\mathbb{Z}$**, in prof :** $\mathbb{Z}$): Que reordena las señales de la reunión según el tono de voz del hablante (de mayor a menor).

Ejercicio 8. **proc silencios(in s:** ***señal*****, in prof:** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$**, in umbral:** $\mathbb{Z}$**, out intervalos:** $seq\langle intervalo\rangle$): Que dada una señal, determina los silencios (pares de indices de inicio y final del silencio). Diremos que hay un silencio si en cierto momento de la señal, todo valor absoluto de la misma no pasa cierto umbral por al menos 0.1 segundos. Tomar en cuenta que antes del inicio y después del final de los silencios encontrados, no puede seguir habiendo silencio y que los intervalos no pueden estar repetidos.

Ejercicio 9. **proc hablantesSuperpuestos(in r:** ***reunion*****, in prof:** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$**, in umbral:** $\mathbb{Z}$**, out result:** Bool): Que dice si en algún momento hay más de una persona hablando (tomando en cuenta la definición de silencio del punto anterior).

Ejercicio 10. **proc reconstruir(in s:** ***señal*****, in prof:** $\mathbb{Z}$**, in freq:** $\mathbb{Z}$**, out result:** ***señal***): Dada una señal de audio, es posible que durante el proceso de muestreo haya fallado, con lo que el sistema de adquisición puso un valor cero en algunas posiciones. En estos casos, la señal debe ser reconstruida de la siguiente manera: si en una posición hay un valor faltante, se debe completar con el promedio de los dos valores no nulos de la secuencia original más cercanos. Sin embargo, estos faltantes no pueden tener más de 4 muestras vacías, ya que una reconstrucción por este método perjudicaría la señal original. Por otro lado, también se debe tener en cuenta la naturaleza sinusoidal de la señal de audio, que posee necesariamente pasajes por ceros. En estos casos, si la muestra es cero, pero corresponde a un valor esperado de la señal, no debe reemplazarse.

Ejercicio 11. **proc filtradoMediana(inout s: *señal*, in R: $\mathbb{Z}$, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$)**: El filtrado mediano se aplica para poder eliminar de la señal valores propios de ruido tipo spikes, de muy corta duración pero de gran potencia. Dado un filtro de largo $F = 2 * R + 1$ muestras, el filtrado de una posición i dentro de la señal consiste en tomar F muestras centradas en i, y reemplazarla por el valor mediano de este conjunto de F muestras. La señal de salida tiene el mismo largo que la de entrada. Los valores extremos de la señal de entrada no son filtrados ya que no es posible obtener la ventana de F muestras con las R muestras pasadas y las R futuras. Para este ejercicio tomaremos 2 y 4 como posibles valores de R. El filtrado se realiza sobre la misma señal de entrada.

Funciones C++

La declaración de las funciones a implementar es la siguiente:

```
bool esSenial(vector<int> const& s, int prof, int freq);
bool seEnojo(senial const& s, int umbral, int prof, int freq);
bool esReunionValida(reunion const& r, int prof, int freq);
void acelerar(reunion& r, int prof, int freq);
void ralentizar(reunion& r, int prof, int freq);
vector<hablante> tonosDeVozElevados(reunion const& r, int freq, int prof);
void ordenar(reunion& r, int freq, int prof);
vector<pair<int,int> > silencios(senial const& s, int prof, int freq, int umbral);
bool hablantesSuperpuestos(reunion const& r, int prof, int freq, int umbral);
senial reconstruir(senial const& s, int prof, int freq);
void filtradoMediana(senial& s, int R, int prof, int freq);
```

Donde declaramos las siguientes estructuras de datos

```
typedef vector<int> senial;
typedef int hablante;
typedef vector<pair<senial, hablante>> reunion;
typedef pair<int, int> intervalo;
```

Uso de pair Vamos a utilizar el container pair que pertenece a la librería estándar del C++ y está definido en el header **¡utility¿**. Este es un container que puede poner juntos dos elementos de cualquier tipo. Para asignar u acceder al primero se utiliza la propiedad *first*, y para el otro... *second*. En el siguiente ejemplo[1], veremos varias maneras de declarar, asignar e imprimir los valores de containers pair.

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

int main()
{
    pair <int, char> PAIR1 ;
    pair <string, double> PAIR2 ("GeeksForGeeks", 1.23) ;
    pair <string, double> PAIR3 ;

    PAIR1.first = 100;
    PAIR1.second = 'G' ;

    PAIR3 = make_pair ("GeeksForGeeks␣is␣Best",4.56);

    cout << PAIR1.first << "␣" ;
    cout << PAIR1.second << endl ;

    cout << PAIR2.first << "␣" ;
    cout << PAIR2.second << endl ;

    cout << PAIR3.first << "␣" ;
    cout << PAIR3.second << endl ;

    return 0;
}
```

[1] https://www.geeksforgeeks.org/pair-in-cpp-stl/

6. Especificación

proc esSeñal (in s: $seq\langle\mathbb{Z}\rangle$, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: Bool) {
  Pre $\{True\}$
  Post $\{result = \text{true} \leftrightarrow esValida(s, prof, freq)\}$

  pred esValida (s: *señal*, prof: $\mathbb{Z}$, frec: $\mathbb{Z}$) {
    $frecValida(frec) \land profValida(prof) \land enRango(s, prof) \land duraMasDe(s, frec, 1)$
  }
  pred frecValida (freq: $\mathbb{Z}$) {
    $frec = 10^2$
  }
  pred enRango (s: *señal*, prof: $\mathbb{Z}$) {
    $(\forall\, x : \mathbb{Z})\ x \in s \longrightarrow -2^{(prof-1)} \leq x \leq 2^{(prof-1)} - 1$
  }
  pred profValida (prof: $\mathbb{Z}$) {
    $prof = 8 \lor prof = 16 \lor prof = 32$
  }
  pred duraMasDe (s: *señal*, freq: $\mathbb{Z}$, seg: $\mathbb{R}$) {
    $|s| >= frec * seg$
  }
}

proc séEnojó? (in s: *señal*, in umbral: $\mathbb{Z}$, in prof : $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: Bool) {
  Pre $\{esValida(s, prof, freq) \land umbralValido(umbral)\}$
  Post $\{result = \text{true} \leftrightarrow (\exists subSenial : señal)(\exists i : \mathbb{Z})(\exists j : \mathbb{Z}) 0 \leq i < j < |s| \land subSec(s, i, j) = subSenial \land duraMasDe(subSenial, freq, 2) \land superaUmbral(subSenial, umbral)\}$

  pred superaUmbral (s: *señal*, umbral: $\mathbb{Z}$) {
    $tono(s) > umbral$
  }
  aux tono (s: *señal*) : $\mathbb{R} = \sum_{i=0}^{|s|-1} abs(s[i])/|s|$;
  pred umbralValido (umbral: $\mathbb{Z}$) {
    $umbral > 0$
  }
}

proc esReuniónVálida (in r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: Bool) {
  Pre $\{True\}$
  Post $\{result = \text{true} \leftrightarrow reunionValida(r, prof, freq)\}$

  pred reunionValida (r: *reunion*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {
    $|r| > 0 \land esMatriz(r) \land senialesValidas(r, prof, frec) \land hablantesDeReunionValidos(r, prof, frec)$
  }
  pred esMatriz (r: $seq\langle señal \times \mathbb{Z}\rangle$) {
    $(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L |r[a]| = |r[0]|$
  }
  pred señalesValidas (r: *reunion*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {
    $(\forall i : \mathbb{Z})(\ 0 \leq i < |r| \longrightarrow_L esValida((r[i])_0, prof, freq))$
  }
  pred hablantesDeReunionValidos (r: *reunion*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {
    $(\forall i : \mathbb{Z})(0 \leq i < |r| \longrightarrow_L (0 \leq r[i]_1 < |r| \land$
    $(\forall j : \mathbb{Z})(0 \leq j < |r| \land_L r[j]_1 = r[i]_1 \longrightarrow i = j)))$
  }
}

proc acelerar (inout r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$) {
  Pre $\{r = r_0 \land reunionValida(r, prof, freq) \land longitudesValidas(r, freq)\}$
  Post $\{|r| = |r_0| \land_L reunionAcelerada(r, r_0)\}$

  pred longitudesValidas (r: reunion, in freq: $\mathbb{Z}$) {

[2]La frecuencia está medida en Hz. 1kHz (1000 muestras por segundo) equivale a 1000 Hz

$(\forall i : \mathbb{Z})\ 0 \leq i < |r| \longrightarrow_L \left\lfloor \frac{|r[i]_0|}{2} \right\rfloor \geq freq * 1000)$

}

`pred reunionAcelerada` (r: *reunion*, r_0: reunion) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L señalAcelerada((r[a])_0, (r_0[a])_0) \wedge mismoHablante((r[a])_1, (r_0[a])_1)$

}

`pred mismoHablante` (h: *hablante*, h1: *hablante*) {

$h = h_1$

}

`pred señalAcelerada` (s: *señal*, s_0: *señal*) {

$|s| = \left\lfloor \frac{|s_0|}{2} \right\rfloor \wedge_L\ (\forall i : \mathbb{Z})\ 0 \leq i < |s| \longrightarrow_L s[i] = s_0[2 * i + 1]$

}

}

`proc ralentizar` (inout r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$) {

`Pre` $\{r = r_0 \wedge reunionValida(r, prof, freq)\}$

`Post` $\{|r| = |r_0| \wedge_L reunionInterpolada(r, r_0)\}$

`pred reunionInterpolada` (r: *reunion*, r_0: *reunion*) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L senialInterpolada((r[a])_0, (r_0[a])_0) \wedge mismoHablante((r[a])_1, (r_0[a])_1)$

}

`pred senialInterpolada` (s: *señal*, s_0: *señal*) {

$|s| = 2 * |s_0| - 1 \wedge_L$

$((\forall i : \mathbb{Z})\ (0 \leq i < |s| - 1 \wedge i \ \text{ mód } \ 2 = 1 \longrightarrow_L \left\lfloor \frac{s[i-1] + s[i+1]}{2} \right\rfloor = s[i])) \wedge$

$(\forall i : \mathbb{Z}) 0 \leq i < |s_0| \longrightarrow_L s_0[i] = s[2 * i]$

}

}

`proc tonosDeVozElevados` (in r: *reunion*, in freq: $\mathbb{Z}$, prof : $\mathbb{Z}$, out hablantes: $seq\langle hablante\rangle$)) {

`Pre` $\{reunionValida(r, prof, freq)\}$

`Post` $\{(\forall h : hablante)(h \in hablantes \longleftrightarrow$

$(\exists t : señal \times hablante) pertenece(t, r) \wedge t_1 = h \wedge tieneElTonoMasElevado(r, t, h, prof, freq)\}$

`pred tieneElTonoMasElevado` (r:*reunion*, t: *señal* $\times$ *hablante*, h: *hablante*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {

$(\forall x : \mathbb{Z})\ 0 \leq x < |r| \longrightarrow_L tono((r[x])_0) <= tono(t_0)$

}

}

`proc ordenar` (inout r:*reunion*, in freq: $\mathbb{Z}$, in prof : $\mathbb{Z}$) {

`Pre` $\{r = r_0 \wedge reunionValida(r_0, prof, freq)\}$

`Post` $\{|r| = |r_0| \wedge_L reunionOrdenada(r, r_0)\}$

`pred reunionOrdenada` (r: *reunion*, r_0: reunion) {

$mismaReunion(r, r_0) \wedge ordenada(r)$

}

`pred mismaReunion` (r: *reunion*, r_0: reunion) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L r[a] \in r_0) \wedge (\forall a : \mathbb{Z})\ 0 \leq a < |r_0| \longrightarrow_L r_0[a] \in r)$

}

`pred ordenada` (r: *reunion*) {

$(\forall a : \mathbb{Z})\ 0 < a < |r| \longrightarrow_L tono(r[a-1]_0) \geq tono(r[a]_0)$

}

}

`proc silencios` (in s: *señal*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, in umbral: amplitud, out intervalos: $seq\langle intervalo\rangle$) {

`Pre` $\{esValida(s, prof, freq) \wedge umbralValido(umbral)\}$

`Post` $\{sinRepetidos(intervalos) \wedge_L$

$(\forall i : \mathbb{Z}) 0 \leq i < |intervalos| \leftrightarrow esSilencio(s, intervalos[i], freq, umbral)\}$

`pred sinRepetidos` (intervalos : $seq\langle intervalo\rangle$) {

$\neg hayRepetido(intervalos)$

```
    }
    pred hayRepetido (intervalos : seq⟨intervalo⟩) {
        (∃i : ℤ)(∃j : ℤ)(0 ≤ i < |intervalos| ∧ 0 ≤ j < |intervalos| ∧ i ≠ j ∧ intervalos[i]_0 = intervalos[j]_0 ∧
        intervalos[i]_1 = intervalos[j]_1)
    }
    pred esSilencio (s: señal, inter : intervalo, freq: ℤ, umbral: ℤ) {

        intervaloEnRango(s, inter) ∧_L duraMasDe(subSec(s, inter_0, inter_1), frec, 0,1) ∧
        intervaloDeSilencio(s, inter, umbral) ∧
        noHayMasGrandeQueLoContiene(s, inter, umbral)

    }
    pred intervaloEnRango (s:señal, inter : intervalo) {
        0 ≤ inter_0 < inter_1 < |s|
    }
    pred intervaloDeSilencio (s: señal, inter : intervalo, umbral: ℤ) {

        (∀k : ℤ) inter_0 ≤ k ≤ inter_1 ⟶_L abs(s[k]) < umbral

    }
    pred noHayMasGrandeQueLoContiene (s: señal, inter : intervalo, umbral: ℤ) {

        inter_0 ≠ 0 ⟶_L abs(s[inter_0 − 1]) ≥ umbral ∧
        inter_1 ≠ |s| − 2 ⟶_L abs(s[inter_1 + 1]) ≥ umbral

    }
}

proc hablantesSuperpuestos (in r: reunion, in prof: ℤ, in freq: ℤ, in umbral: ℤ, out result: Bool) {
    Pre {esReunionValida(r, prof, freq) ∧ umbralValido(umbral)}
    Post {result = true ↔ (∃h1, h2 : hablante) 0 ≤ h1 < h2 < |r| ∧_L ¬seRespetan(r, h1, h2, freq, umbral)}

    pred seRespetan (r: reunion, h1: hablante, h2: hablante, freq: ℤ, umbral: ℤ) {
        (∀i : ℤ) 0 ≤ i < |r[h1]| ⟶_L
        (¬haySilencioQueLoContiene(r[h1])_0, i, freq, umbral) →
        haySilencioQueLoContiene(r[h2])_0, i, freq, umbral))
    }
    pred haySilencioQueLoContiene (s : señal, i : ℤ, freq : ℤ, umbral : ℤ) {

        (∃inter : intervalo)(intervaloEnRango(s, inter) ∧_L
        inter_0 ≤ i ≤ inter_1 ∧
        esSilencio(s, inter, freq, umbral)
    }
}

proc reconstruir (in s: señal, in prof: ℤ, in freq: ℤ, out result: señal) {
    Pre {esValida(s, prof, freq) ∧_L s[0] ≠ 0 ∧ s[|s| − 1] ≠ 0 ∧ puedeReconstruirse(s)}
    Post {esReconstruida(s, result)}

    pred puedeReconstruirse (s: señal) {
        (∀i : ℤ)(∀j : ℤ)(0 ≤ i < j < |s| ∧_L s[i] ≠ 0 ∧ s[j] ≠ 0 ∧ todosCeros(subSeq(s, i + 1, j)) ⟶_L distancia(i, j) < 5)
    }
    pred todosCeros (s: señal) {
        (∀i : ℤ)0 ≤ i < |s| →_L s[i] = 0
    }
    pred esReconstruida (s: señal, sRec: señal) {
        |s| = |sRec| ∧_L
        (∀i : ℤ)(0 ≤ i < |s| →_L
        ((s[i] = 0 ∧ reconstruirPosicionSiCorresponde(s, sRec, i)) ∨
        (s[i] ≠ 0 ∧ sRec[i] = s[i]))
    }
    pred reconstruirPosicionSiCorresponde (s: señal, sRec:señal, i: ℤ) {
        (esPasajePorCero(s, i) ∧ s[0] = 0) ∨ (¬esPasajePorCero(s, i) ∧ esValorEnPosicion(s, sRec[i], i))
```

```
    }
    pred esPasajePorCero (s: señal, i: ℤ) {
        signo(s[i − 1]) ∗ signo(s[i + 1]) = −1
    }
    pred esValorEnPosicion (s: señal, valor: ℤ, i: ℤ) {
        (∃j : ℤ)(∃k : ℤ)(masCercanosNoNulos(s, i, j, k) ∧L valor = ⌊(s[k]+s[j])/2⌋)
    }
    pred masCercanosNoNulos (s: señal, i: ℤ, j: ℤ, k: ℤ) {
        0 ≤ j < i < k < |s| ∧L s[j] ≠ 0 ∧ s[k] ≠ 0 ∧ distancia(j, k) ≤ 5 ∧
        (∀l : ℤ)(0 ≤ l < |s| ∧L s[l] ≠ 0 ⟶ distancia(i, l) ≥ max(distancia(i, j), distancia(i, k)))
    }
    aux distancia (j: ℤ, i: ℤ) : ℤ = |i − j| ;
    aux signo (i: ℤ) : ℤ = if x < 0 then −1 else (if x > 0 then 1 else 0 fi) fi ;
}

proc filtradoMediana (inout s: señal, R: ℤ, in prof: ℤ, in freq: ℤ) {
    Pre {esValida(s, prof, freq) ∧ s = s0 ∧ R = 2 ∨ R = 4}
    Post {esFiltrada(s0, s, R)}
    pred esFiltrada (s: señal, sFilt: señal, R: ℤ) {
        |s| = |sFilt| ∧ (∀i : ℤ)(0 ≤ i < |s| →L
        (coincidenExtremos(s, sFilt, i, R) ∨L esValorFiltrado(s, sFilt, i, R)))
    }
    pred coincidenExtremos (s: señal, sFilt: señal, i: ℤ, R: ℤ) {
        (i < R ∨ i ≥ |s| − R) ∧ s[i] = sFilt[i]
    }
    pred esValorFiltrado (s: señal, sFilt: señal, i: ℤ, R: ℤ) {
        (∃w : señal)|w| = 2∗R+1∧mismosValores(w, subSeq(s, i−R, i+R+1))∧valoresOrdenados(w)∧L sFilt[i] = w[R]
    }
    pred mismosValores (s: señal, q: señal) {
        (∀v : ℤ)v ∈ s ⟶L #apariciones(s, v) = #apariciones(q, v)
    }
    pred valoresOrdenados (s: señal) {
        (∀i : ℤ)0 < i < |s| ⟶L s[i − 1] ≤ s[i]
    }
}
```