Enginyeria Tècnica en Informàtica de Sistemes
Treball fi de carrera
Universitat Oberta de Catalunya

NORMALITZACIÓ D'UN VOLUM D'ARXIUS EN FORMAT MP3

Autor: **Albert Domènech Lozano**
Consultor: **Antonio Bonafonte Cávez**

28 de Juny del 2002

ÍNDEX

1. Introducció

1.1. Introducció a MP3

El nom d'aquest format prové del format de compressió d'àudio anomenat MPEG Layer III.

L'algorisme de compressió d'àudio MPEG (*Motion Picture Experts Group*) és un estàndard per a la compressió d'àudio d'alta fidelitat de la ISO (*International Organization for Standardization*). MPEG ofereix tres nivells de compressió, cadascún amb incrementant la complexitat i aportant una millor qualitat de so. El MPEG Layer III (MP3) és l'esquema més complex i aporta la millor qualitat de so entre les tres capes (*layers*). En aquest layer, amb diferents taxes de mostreig podem aconseguir diferents qualitats de so.

Qualitat de so	Amplada de banda	Mode	Taxa de mostreig	Compressió
So telefònic	2,5 KHz	Mono	8 Kbps	96:1
Millor que ona curta	4,5 KHz	Mono	16 Kbps	48:1
Millor que ràdio AM	7,5 KHz	Mono	32 Kbps	24:1
Semblant a ràdio FM	11 KHz	Stereo	56...64 Kbps	26...24:1
Quasi CD	15 KHz	Stereo	96 Kbps	16:1
CD	>15 KHz	stereo	112...128 Kbps	14...12:1

Figura 1: Esquema de les qualitats de so que es poden obtenir en MPEG Layer III

Sense compressió, les senyals d'àudio digitals consten típicament de trames de 16 bits emmagatzemades en un taxa de mostreig de més de dues vegades de l'amplada de banda (*criteri de Nyquist*). Per tant, ens trobem amb, per exemple, més de 1400 Mbits per representar només un segon de música estèreo amb qualitat CD. Fent servir codificació MPEG, podem comprimir les dades de so originals del CD en un factor de 12 sense perdre qualitat de so.

La rao més important per a què l'algorisme de compressió d'àudio MPEG Layer III pugui comprimir senyals d'àudio digital efectivament sense pèrdues apreciables és degut a l'ús de les tècniques de quantificació i de codificació per entropia. La quantificació elimina parts auditives irrelevants sense perdre la qualitat de so i explotant les propietats perceptives del sistema auditiu humà. L'eliminació d'aquestes parts irrelevants resulta en una distorsió inaudible. La codificació per entropia és un mètode de codificació sense pèrdues que codifica les dades quantificades per a minimitzar l'entropia dels valors quantificats de la senyal d'àudio, d'aquesta manera s'aconsegueix l'objectiu de compressió sense perdre qualitat.

Aquestes parts auditives irrelevants que s'eliminen, esmentades anteriorment, fan que la compressió pugui reduïr el tamany de l'arxiu original en una taxa de fins a 12:1, sense produir-se una pèrdua de la qualitat de so. Aquestes parts irrelevants s'elegeixen per les propietats perceptives del sistema auditiu humà i es basen en una tècnica anomenada *emmascarament auditiu*[1].

Per una altra banda, en aquest projecte es tracta la normalització dels arxius MP3.

En la pràctica, ens podem trobar que diferents cançons no tenen la mateixa sonoritat. Això és degut, moltes vegades, a que aquestes cançons no tenen el mateix volum. Aquí és on entra a escena la normalització. La normalització és un procés que analitza els nivells de potència de les diferents parts que formen d'aquests arxius de música. Concretament analitza les diferents components freqüencials que el formen per a poder calcular un factor de normalització, a partir del qual, determinades components freqüencials, les que així ho necessitin, o totes les components freqüencials d'arxius, poden arribar als nivells de potència requerits per a poder obtenir el resultat buscat.

L'anàlisi de les components freqüencials es dur a terme mitjançant una anàlsi de pics de les diferents components i un anàlisi estadístic per a estudiar i determinar la sonoritat amb la que la cançó "sona" per l'oïda humana, la manera de percebre-la. Com es pot comprovar, aquest anàlisi es pot complicar fins a límits insospitats, depenent de l'exigència de l'usuari final.

De totes maneres, el resultat buscat, en tots aquests casos, és la sonoritat uniforme en un volum de diferents arxius de música.

1.2. Introducció al projecte

Aquest projecte s'ha organitzat d'una manera lògica, definint en un primer moment l'objectiu del mateix, junt amb la manera de treballar en aquest projecte per a assolir l'objectiu. Posteriorment ja s'inicia el projecte propiament dit.

En aquesta segona part s'explicarà el sistema i el format d'aquest sistema de compressió d'arxius de so. Començant pel funcionament del codificador MP3 i acavant per la definició del format en sí. En aquesta part sempre s'introduirà, a partir de les definicions desenvolupades, referències en les parts del codi font on es duen

[1] És un fenòmen que es dóna quan hi ha senyals de baix nivell (*emmascarades*) molt juntes en freqüència a una senyal d'alt nivell (*emmascarador*). Llavors es crea automàticament un *llindar d'emmascarament* el qual engloba aquestes freqüències de baix nivell i fa que siguin inaudibles. Aquestes senyals *emmascarades* poden ser senyals de baix nivell, restes de distorsions, errors de transmissió... Les senyals, el nivell de les quals no traspassin el *llindar en silenci* també resulten inaudibles. Aquest llindar varia depenent de la freqüència en la que ens trobem.

a terme rutines relaciones amb l'explicació que s'està descrivint. Això ens donarà, a part de la visió teòrica que vol introduïr aquesta part, una visió pràctica i centrada en l'aplicació objecte d'aquest projecte. També s'ha intentat descriure el codificador i el format MPEG Layer III d'una manera compacte i comprensible donat el caràcter complex que té aquest codificador i format.

Després d'aquesta visió teòrica es passarà a explicar la resolució pràctica per a poder dur a terme l'objectiu del Treball Fi de Carrera (implementació i metodologia).

S'acavarà desenvolupant una conclusió al que ha estat el Treball Fi de Carrera: visió pràctica d'ell, utilitats,... Per després, donar una opinió de les possibles modificacions i activitats futures per a dur a terme una millora en el rendiment.

Finalment es referenciaràn les fonts (bibliografia) d'on s'ha extret informació d'utilitat per a poder dur a terme aquest projecte.

A l'annex final s'adjuntaràn les fonts de l'implementació de l'aplicatiu realitzat.

2. Objectius

2.1. Objectius del projecte

La normalització d'arxius MP3 es pot dividir en vàries parts ben diferenciades:

- Primerament, és necessari disposar d'un programa que ens permeti poder fer una interpretació i manipulació dels arxius MP3 a nivell de trama. S'ha d'aprofundir en el treball d'interpretació a nivell de trama per a poder extreure la representació de les components frecuencials de les trames (representen la forma del so a la que fan referència) que han estat codificades en l'arxiu MP3 a través del sistema de compressió MPEG Layer III.

- En una segona part, s'ha de tractar aquesta informació extreta de les trames (les components freqüencials) d'un volum d'arxius MP3. Aquest tractament consistirà en un anàlisi i comparació dels nivells de potència de les trames de varis arxius per a poder calcular un factor de normalització que ens aporti una amplificació/atenuació de la sonoritat de l'arxiu.

- Per últim, s'ha de definir una rutina per a poder modificar correctament la guanyança global de cada trama a partir del factor de normalització, calculat en la part anterior, i que resultarà en un increment/disminució de la guanyança en les trames.

La divisió en aquestes dues parts en assegura el correcte treball per arribar a la normalització d'arxius MP3.

Aquest Treball Fi de Carrera té com a objectiu a assolir, la primera i tercera de les parts descrites anteriorment. Per tant, en aquest projecte, es busca assolir els objectius de:

- Tractament del format d'arxius MP3 per a realitzar una extracció de components freqüencials, i
- Modificació de la guanyança global de les trames donat una valor donat (factor de normalització).

2.2. Treball i implementació

El treball de realització d'aquest projecte s'ha dividit en dues fases diferents: una d'investigació i una altra d'implementació.

En la fase d'investigació s'ha realitzat un estudi exhaustiu sobre el sistema de compressió MPEG Layer III per a poder observar i entendre el seu funcionament, i també sobre el format MP3 per a entrar a un nivell físic i lògic de l'organització d'aquests arxius. D'aquesta manera s'ha pogut assolir els coneixements teòrics sobre el sistema per a la seva correcta aplicació posterior en el projecte. D'una manera paral·lela s'ha integrat, també, coneixements en la tècnica de normalització d'arxius de so per a conèixer els requisits i les funcions necessàries per a poder realitzar-la. A partir d'aquestes dues vertents en l'investigació s'han pogut assolir uns coneixements globals sobre el projecte. Posteriorment s'ha aprofundit en la primera de les vertents per a poder realitzar la part, en concret, objecte de realització en aquest projecte (la interpretació i manipulació d'arxius MP3).

En la fase d'implementació s'ha posat a la pràctica els coneixements assolits en la fase anterior per a poder acomplir l'objectiu del projecte. Concretament, s'ha posat en pràctica els coneixements en el format MP3 per a poder realitzar la interpretació i manipulació de les trames que formen part dels arxius MP3, mitjançant rutines estructurades convenientment en les fases que es necessiten per a poder interpretar els arxius. Tot això per a arribar a la recollida d'informació de les components frecuencials de les trames i a la correcta modificació de la guanyança global d'aquestes.

3. MPEG Layer III

3.1. El codificador MP3

A continuació es descriurà la funcionalitat del codificador d'àudio MPEG Layer III. El codificador el podem observar gràficament en la següent diagrama de blocs:

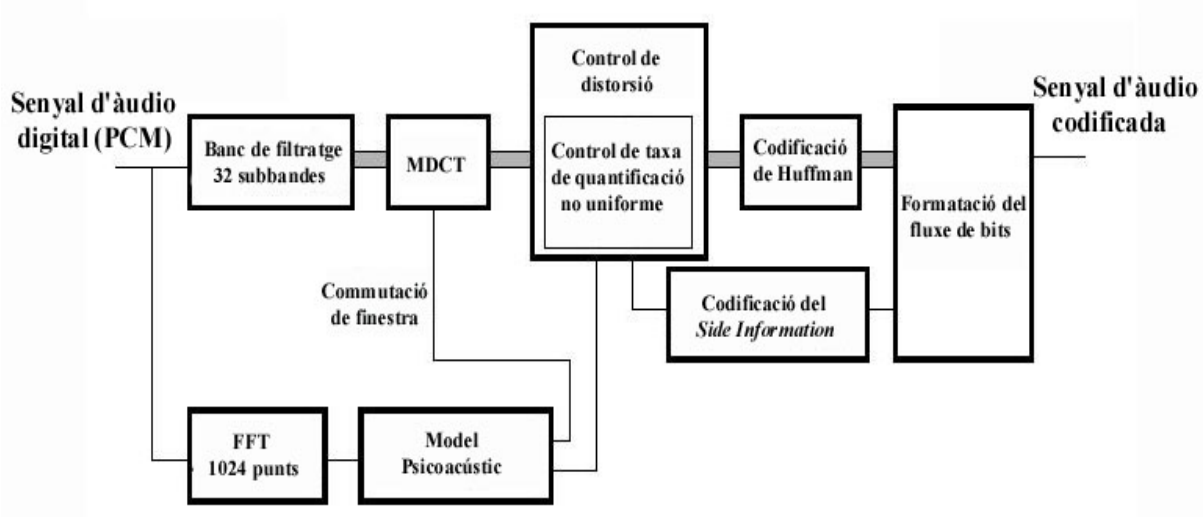

Figura 2: Diagrama de blocs del codificador MPEG Layer III

La senyal àudio d'entrada que prové d'un únic canal d'una senyal PCM passa a través d'un banc de filtratge polifàsic (*polyphase filter bank*). Aquest banc de filtratge divideix la senyal d'entrada en 32 subbandes de freqüència equidistants. Després del procés, les mostres en cada subbanda estàn encara en el domini temporal.

Llavors es fa servir una transformada discreta de cosinus modificada (*MDCT*) per tal de mapejar les mostres en cada subbanda en un domini freqüencial, d'aquest manera es genera una millor resolució espectral. A causa de l'inevitable intercanvi entre la resolució temporal i freqüencial, Layer III especifica dues llargades de bloc MDCT diferents, una llarga de 36 mostres i una curta de 12 mostres. Els blocs curts milloren la resolució temporal per a poder fer front als transients.

Paral·lelament, la senyal d'entrada després de la transformació *FFT*[1]passa a través d'un model psicoacústic que determina la proporció d'energia de la senyal en el llindar d'emmascarament per a cada subbanda. El model psicoacústic és la clau per al gran rendiment del codificador MPEG. La feina d'aquest model psicoacústic és analitzar la senyal àudio d'entrada i determinar on es pot emmascarar l'espectre de soroll de quantificació i el que es pot extendre. El codificador utilitza aquesta informació per decidir com representar millor la senyal d'àudio d'entrada amb un nombre limitat de bits de codificació.

El bloc de control de distorsió fa servir la proporció senyal/màscara (*SMR*) del model psicoacústic per decidir com assignar el nombre total de codis de bits disponibles per la quantificació dels senyals de subbanda per a minimitzar l'audibilitat del soroll de quantificació. Les mostres quantificades de subbanda són llavors codificades amb l'algorisme de Huffman per a decrementar l'entropia de les mostres.

Finalment, el bloc final agafa les mostres de les subbandes codificades amb l'*algorisme de Huffman* i el *Side Information* (definits en el proper punt) i les posa dins del fluxe de bits empaquetat d'acord amb l'estàndard MPEG d'àudio.

El codificador MPEG Layer III té tot un seguit de mètodes característics, que formen part del procés, com poden ser:

- *Reducció d'àlias*, que és un mètode de processament dels valors del MDCT per eliminar redundàncies causades pel solapament de bandes del banc de filtratge.
- *Quantificació no-uniforme*. El quantificador augmenta l'entrada a tres quarts de potència abans de la quantificació per a proveïr un percentatge de senyal-soroll més consistent.
- *Codificació per entropia dels valors*. Utilitza codis de Huffman per codificar les mostres quantificades fer una millor compressió de les dades.
- *Bit de reserva*. Alguns passatges d'una peça musical no poden ser codificats a una tasa donada sense alterar la qualitat musical. Llavors s'usen aquests bits de reserva per actuar com a buffer emmagatzemant els passatges que s'han de codificar a una tasa inferior.
- *Localització de soroll*. El codificador utilitza un bucle iteratiu de localització de soroll. En aquest bucle, els quantificadors són variats de posició, i la quantificació del soroll resultant és calculada i especificada localitzada a cada subbanda.

A partir de la visió donada sobre el codificador d'àudio MPEG Layer III podrem entendre molt més bé el treball en aquest projecte, ja que s'implementa un part del decodificador que, justament, executarà els mateixos blocs que el codificador però a la inversa. D'aquesta manera les funcions que haurà d'anar assolint el decodificador d'àudio MPEG Layer III seràn les següents:

[1] Fast Fourer Transformation. És un algoritme ràpid per a representar una transformada discreta de Fourier (una tranformada ortogonal).

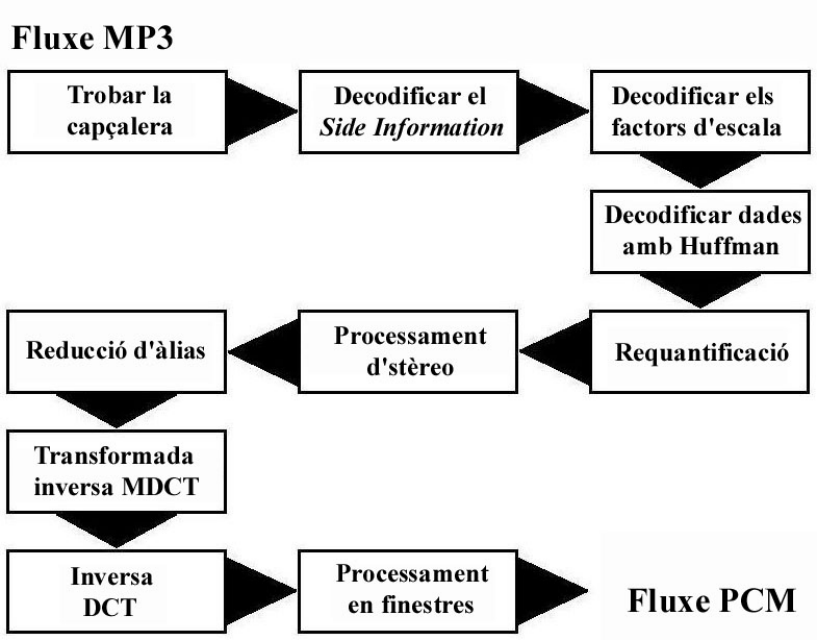

Figura 3: Esquema en part del procés de decodificació d'àudio MPEG Layer III

En el nostre cas, implementarem en diferents rutines els 5 primers blocs de l'esquema anterior. Això és:

1. Trobar la capçalera de cada trama, implementada en la clase *FrameHeader*,
2. Decodificar el Side Information, implementada en la classe *SideInfo*,
3. Decodificar els factors d'escala, implementada dins la classe *MainData*,
4. Decodificar les dades codificades amb l'algorisme de Huffman, implementada dins la classe *MainData*, i
5. Quantificació inversa de les mostres, també implementada dins la classe *MainData*.

Per una altra banda, s'ha detenir en compte que MPEG Layer III dóna suport per a dos modes de codificació bàsics (tots dos suportats en l'implementació d'aquest projecte):

- Codificació amb taxa variable (*Variable rate coding*). Aquesta codificació significa que el codificador pot anar canviant el bitrate en cada trama depenent de les demandes psicoacústiques. Estàn permesos tots els *bitrate*. La decodificació dels fluxes de bits només poden ser realitzats "sota demanda", és a dir, el decodificador comunica al servidor la quantitat de dades que es necessita per a continuar decodifcant. És impossible la transmissió en una taxa constant per aquests fluxes de bits.

- Codificació amb taxa constant (*Constant rate coding*). Aquest tipus de codificació es fa servir per transmissions en una taxa constant. Per a

transmissions a taxa constant, Layer III suporta la commutació de els *bitrates* entre valors propers d'índex de *bitrate*. Aquesta característica es fa servir per diferents propòsits:

1. L'emulació de bitrates intermitjos sense utilitzar "format lliure". Per exemple, un *bitrate* de 60 Kbps pot ser aconseguit per la contínua commutació entre 64 Kbps i 56 Kbps.

2. El mode d'operació asíncron. El Layer III permet treballar amb rellotges asíncrons de fluxes de bits a la freqüència de mostratge de les dades d'entrada d'àudio. L'asincronisme s'intercepta per la commutació de *bitrates*. En el decodificador, la freqüència de mostratge a l'entrada del codificador pot èsser reconstruïda. La commutació asíncrona de *bitrates* requereix l'ús almenys de tres *bitrates* propers. Per exemple, per realitzar l'operació asíncrona a un *bitrate* de 64 Kbps, els *bitrates* 56 Kbps i 80 Kbps poden ser utilitzats igualment per a ajustar la taxa de les dades.

Un decodificar completament compatible amb el definit per la ISO ha de suportar la codificació de taxa variable així com els dos tipus de codificació de taxa constant.

3.2. El format MP3

Els arxius MP3 estàn segmentats en milers de trames, cadascuna conté el valor d'una fracció de segon de les dades d'àudio, preparada per a ser reconstruïda pel decodificador.

Insertat al principi de cada trama hi ha la capçalera (*header frame*), que emmagatzema 32 bits de dades relacionades amb les dades de la trama que segueixen a la capçalera. L'estructura de la capçalera és la següent:

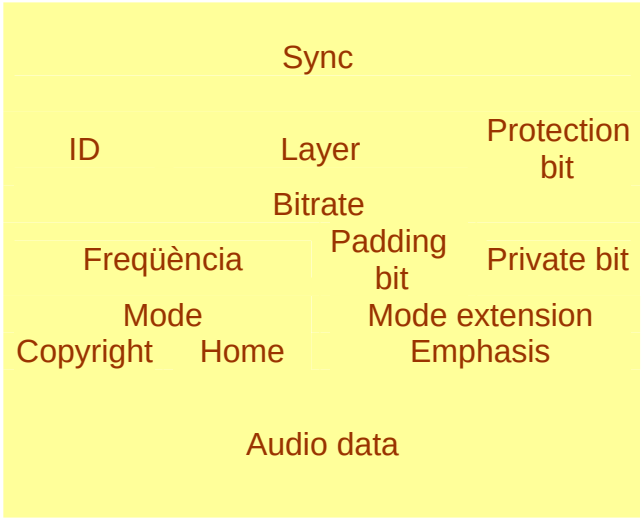

Figura 4: Estructura de la capçalera de cada trama

La capçalera comença amb el bloc *sync* (paraula de sincronisme) que consisteix en 11 bits. La paraula de sincronisme permet als reproductors poder buscar la primera ocurrència d'una trama vàlida, la qual cosa és útil per a la difusió MP3 (*MP3 broadcasting*), per moure's ràpidament des d'una part de la reproducció a una altra, i per saltar-se tags (ID3) o dades que poden estar al principi. De totes maneres no és suficient, per un reproductor, el fet de trobat la paraula de sincronisme per assumir que es tracta d'un arxiu MP3 vàlid, ja que es podria trobar aquest patró en una arxiu binari qualsevol. Per tant, és també necessari, pel decodificador, comprovar la validesa de les altres dades de la capçalera.

Seguidament a la paraula de sincronisme hi ha el bit *ID*, que especifica si la trama ha estat codificada en MPEG-1 o MPEG-2. El segueixen dos bits de *Layer*, que especifica si la trama és Layer I, Layer II, Layer III o no està definida. Si el *Protection bit* no és igual a 1, un checksum de 16 bits serà insertat abans del principi del *Audio data*.

El camp de *Bitrate* especifica la taxa de mostreig de la trama actual, que va seguida d'un especificador de la freqüència d'àudio. El *Padding bit* es fa servir per assegurar-se de que cada trama satisfà els requeriments de la taxa de mostreig exactament. Per exemple, en el cas en que ens trobem en un arxiu en Layer III amb una taxa de mostreig de 128 Kbps i una feqüència de mostreig de 44,1 KHz contindrà trames de 417 bytes i de 418 bytes. Les trames de 417 bytes tindràn el *Padding bit* igual a 1 per a conservar la discrepància.

El camp *Mode* es refereix a l'estat stereo/mono de la trama i permet l'assignació d'opcions de codificació stereo, doble canal i mono. Si s'ha habilitat efectes *joint stereo*[1], el camp *Mode extension* dirà al decodificador, exactament, com tractar-lo; en el cas, per exemple, de si les altes feqüències han estat combinades entre canals.

[1] En algunes configuracions HI-FI de rang mig, hi ha un únic subwoofer; de totes maneres, generalment, no tenim la sensació de que el so provingui d'aquest subwoofer, a l'igual passa amb els altaveus satel·litals. Sabem

El bit de *Copyright* no conté informació en sí de copyright, però fa servir un copyright similar al dels CDs i DATs. Si el bit està activat, oficialment és il·legal copiar el "tall". Si les dades es troben en el seu medi original, el bit de *Home* estarà activat. El *Private bit* per ser utilitzat per aplicacions específiques per accionar procediments personalitzats.

El camp *Emphasis* s'utilitza com a flag, en el cas de que estigui activat en la grabació original. Rarament és utilitzat.

Finalment, el decodificador es mou a través del checksum (si existeix) i sobre el *Audio data* de la trama.

En aquest moment, comença el *Audio data* que està format pel *Side information* i el *Main data*. Mentres el *Main Data* té una llargada sempre variable, el *Side information* té sempre un llargada fixa: 17 bytes en mode canal únic i 32 bytes en els altres modes.

El *Side information* de la trama conté l'informació necessària per a decodificar el *Main Data*. El *Side information* conté informació relacionada amb les taules de Huffman, per utilitzar en el procés de decodificació de Huffman, i informació dels factors d'escala.

En aquesta secció, el *Side information*, hi ha dades per a cada canal en ambdós *granules*[1] (un en MPEG-2 i dos en MPEG-1). El *Side information* inclou dades importants com poden ser el *main_data_end*, que ens indica on s'acava el *Main data*; i el *part2_3_length*, que ens indica la llargada dels factors d'escala i les dades codificades en Huffman (explicat posteriorment).

De la mateixa manera podrem trobar camps que ens aporten informació sobre els factors d'escala, *scfsi* i *scalefac_compress*; valors espectrals, *big_values*; guanyança de les trames, *global_gain* , utilitzada precisament per a poder modificar la guanyança de les trames (objectiu d'una part d'aquest projecte), i *subblock_gain*; taules de codificació de Huffman, *table_select* i *count1table_select*. A part d'aquests camps en podem trobar alguns d'altres que també ens seràn d'utilitat.

Seguidament ens trobarem amb el *Main data*, encara que, no necessariament es troba tot seguit al *Side Information*. La localització del *Main data* quedarà definida pel punter *main_data_end*, que ens indica el seu final, i el principi que vindrà definit pel final del *Main data* anterior. Totes aquestes parts en les que es divideix una trama

---

que l'oïda humana, en freqüències molt baixes i molt altes, no es capaç de localitzar l'orígen espaial dels sons amb precissió. Llavors el format MP3 pot revertir això amb un "truc", fent servir el que s'anomena Intensity Stereo. Algunes freqüències són grabades com a senyals monofòniques seguides per una informació addicional per a resturar mínimament l'espaialitat. La segona eina joint stereo, anomenada Mig/Costat (M/S) stereo, quan els canals dret i esquerre són molt similars, es codifiquen uns canals mig (L+R) i costat (L-R) en comptes del dret i l'esquerre. Això permet reduïr el tamany de l'arxiu final fent servir menys bits pels canals dels costats. En la reproducció, el decodificador MP3 reconstruirà els canals dret i esquerre.

[1] En MPEG Layer III cada granule conté 576 línies de freqüència que contenen la seva pròpia *Side information*.

que forma part d'un arxiu MP3 les podem veure representades més comprensiblement en el següent esquema:

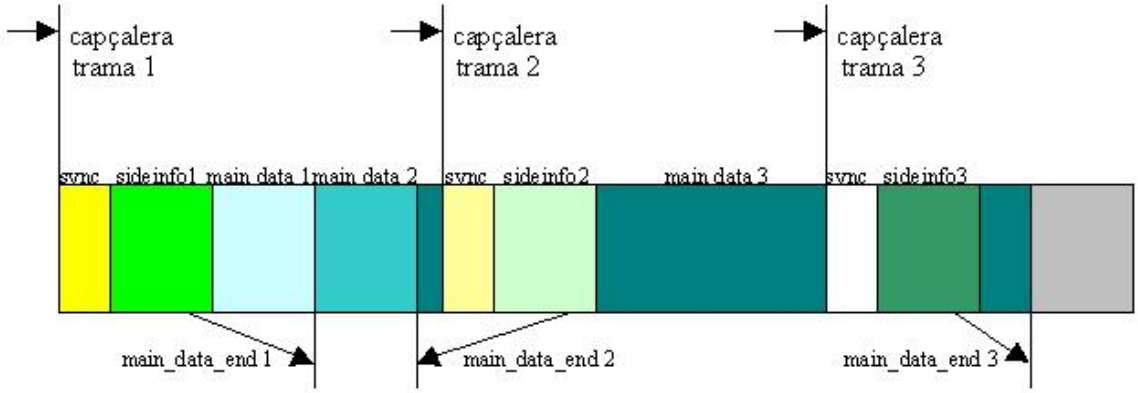

Figura 5: Representació gràfica del format de les trames

El *Main data* consisteix en dos granules, en MPEG 1, o en un granule, en MPEG 2; cadascún conté un canal (en mode canal únic) o dos canals (en els altres modes). Cada canal conté *scalefactors*[1] (factors d'escala) i *Huffman code bits*[2] (dades codificades en l'algorisme de Huffman).

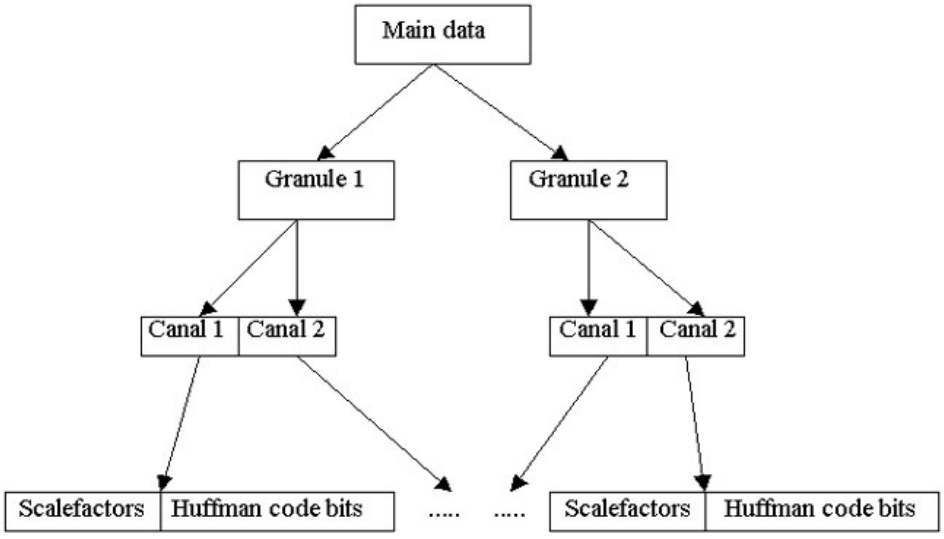

Figura 6: Representació esquemàtica del Main data

[1] Factors pels quals un grup de valors estàn escalats abans de la quantificació.
[2] Són bits codificats amb l'algorisme de Huffman. Aquesta codificació crea codis amb llargada variable per un conjunt de bits (símbol). Els símbols amb més probabilitat tenen codis més curts. Els codis de Huffman tenen la propietat de tenir un prefix únic, d'aquest manera poden ser codificats correctament sense que afecti la seva llargada variable. La decodificació, al realitzar-se amb unes taules de correspondències, es fa molt ràpid. Aquest tipus de codificació permet estalviar la mitjana d'espai ocupat en un 20%.

Per tant, en el *Main data*, es va repetint la parella (*Scalefactors, Huffman code bits*) per a tots els granules i canals.

Un cop representades totes les parelles, apareixen els *ancilliary bits* (bits auxiliars), que també formen part del *Main data* i són bits definibles per l'usuari en un nombre que pot anar variant. Els *ancilliary bits* es situaràn, doncs, entre els últims *Scalefactors* i *Huffman code bits*, que representen l'últim granule/canal, i el principi del *Main data* de la següent trama.

D'aquesta manera, obtindrem per MPEG 1 la seqüència següent dins del Main data:

- Factors d'escala (*Scalefactors*) pel gränul 0 i canal 0.
- Dades codificades amb l'algorisme de Huffman (*Huffman code bits*) pel gränul 0 i canal 0.
- Factors d'escala (*Scalefactors*) pel gränul 0 i canal 1.
- Dades codificades amb l'algorisme de Huffman (*Huffman code bits*) pel gränul 0 i canal 1.
- Factors d'escala (*Scalefactors*) pel gränul 1 i canal 0.
- Dades codificades amb l'algorisme de Huffman (*Huffman code bits*) pel gränul 1 i canal 0.
- Factors d'escala (*Scalefactors*) pel gränul 1 i canal 1.
- Dades codificades amb l'algorisme de Huffman (*Huffman code bits*) pel gränul 1 i canal 1.
- Bits auxiliars (*Ancilliary bits*).

Per MPEG 2 la seqüència serà la següent:

- Factors d'escala (*Scalefactors*) pel gränul 0 i canal 0.
- Dades codificades amb l'algorisme de Huffman (*Huffman code bits*) pel gränul 0 i canal 0.
- Factors d'escala (*Scalefactors*) pel gränul 0 i canal 1.
- Dades codificades amb l'algorisme de Huffman (*Huffman code bits*) pel gränul 0 i canal 1.
- Bits auxiliars (*Ancilliary bits*).

En totes aquestes seqüències es suposa que ens trobem en el cas de que estem tractant trames amb dos canals (*estèreo*), però ens podríem trobar amb trames amb un canal (*mono*), en aquest cas els factors d'escala i les dades codificades amb l'algorisme de Huffman per al segon canal no existirien.

En la decodificació d'aquests bits codifcats amb l'algorisme de Huffman (*Huffman code bits*), tota l'informació necessària per a dur a terme el procés pot ser generada a partir de les anomenades taules de Huffman. En l'implementació

d'aquest projecte s'ha generat unes taules especials (en l'arxiu *huffman.h*, inclòs en l'Annex). El procés que es segueix en la decodificació d'aquests bits és el següent:

1. Decodificar les línies de freqüència per a la primera *regió*[1] (definida en un camp del *Side Information*) en parells, amb l'ajuda de la taula definida en el camp *table_select* del *Side Information*, i única per a cada regió.
2. Decodificar les línies de freqüència per a la segona regió (definida en un camp del *Side Information*) en parells, amb l'ajuda de la taula definida en el camp *table_select* del *Side Information*, i única per a cada regió.
3. Decodificar les línies de freqüència per a la tercera regió (definida en un camp del *Side Information*) en parells, amb l'ajuda de la taula definida en el camp *table_select* del *Side Information*, i única per a cada regió.
4. Decodificar les regions anteriors fins que s'arriba al nombre de parells de línies indicades pel camp *big_values* del *Side Information*.
5. Els bits codificats amb l'algorisme de Huffman restants, són decodificats utilitzant la taula indicada pel camp *count1table_select* del *Side Information*. Aquesta descodificació es dur a terme fins que tots els bits restants han estat decodificats o fins que s'hagin decodificat 576 línies de freqüència.
6. Els bits sobrants, en el cas que existeixin, són descartats.

Aquestes dades codificades amb l'algorisme de Huffman estàn també quantificades. Aquest procés de requantificació es dur a terme a través d'una fórmula que descriu tot el processament des dels valors decodificats de Huffman cap a l'entrada del banc de filtratge. Tots els factors d'escala necessaris estàn continguts dins de la fórmula. Les dades de sortida són reconstruïdes des de les mostres desquantificades.

D'aquesta part del frame es d'on podrem obtenir les mostres que formen part de cada trama, a través de la descodificació i la desquantificació de les dades descrites en els pàrrafs anteriors.

L'estructura dels fluxes MP3 ha d'estar formatejada convenientment i seguint l'estàndard definit anteriorment de manera concisa. A través de les diferents classes que formen part de la implementació (*FrameHeader, SideInfo, MainData*) s'interpreta el format MP3 tal i com s'ha descrit.

[1] Per a afavorir un bon particionament de l'espectre, s'utilitzen les regions per resaltar el funcionament del codificador de Huffman. És una subdivisió de la regió descrita pel camp *big_values* del *Side Information*. El propòsit d'aquesta subdivisió és obtenir un millor robustesa enfront a errors i una millor eficiència en la codificació.

4. Implementació

4.1. Generalitats de la implementació

La implementació duta a terme en aquest Treball Fi de Carrera consta del codi necessari per a assolir els objectius marcats amb anterioritat d'una manera totalment eficient i pràctica.

S'ha intentat generar un codi el més entenedor possible per a fer possible l'intel·ligibilitat de qualsevol part del mateix per a qualsevol persona que no hi hagi estat en contacte amb anterioritat. En aquesta línia, s'ha afegit, en totes les rutines i parts de codi susceptibles a dubtes, uns comentaris explicatius sobre la funció que realitzen per a poder mantenir la correcta interpretació del mateix.

Tot el codi ha estat generat en el llenguatge de programació C/C++, degut a la seva potència en el tractament d'arxius binaris. El IDE utilitzat ha estat el Bloodshed Dev-C++ 4, que està sota una llicència general pública (GNU). El maquinari utilitzat que s'ha utilitzat en les feines de programació, depuració i proves ha estat un ordinador amb processador Intel Pentium-II a 400 MHz amb 256 MB de memòria RAM.

L'execució del programari generat en aquest projecte es pot dur a terme en qualsevol maquinari capaç de reproduïr arxius en format MP3, és a dir, ordinadors que disposin d'un processador a partir de Pentium, amb sistemes operatius Windows 95 o superiors.

La implementació completa del programari generat està inclosa en l'Annex final d'aquest projecte.

4.2. Metodologia utilitzada

Com ja s'ha anat comentant en punts anteriors, la implementació que s'ha fet ha intentat seguir una estructuració completament intel·ligible i organitzada en totes les parts del codi. Juntament amb uns comentaris explicatius en totes les parts del mateix codi.

S'ha utilitzat vàries classes per a representar les diferents fases que s'han de dur a terme en la decodificació d'arxius d'àudio en format MPEG Layer III. Es parteix d'un esquelet format per les parts més visibles de l'interpretació d'aquests arxius:

- Lectura i interpretació de les capçaleres de les trames (*Frame Header*) que formen l'arxiu MP3. La qual cosa també implica la sincronització de les trames i el correcte emmagatzemament dels camps de la capçalera.
- Lectura i interpretació de la part *Side Information* de cadascuna de les trames. Comporta el correcte emmagatzemament i ús posterior dels valors dels seus camps.
- Lectura, interpretació i manipulació de la part *Main Data*. Inclou la correcta l'extracció dels factors d'escala (*scalefactors*) i dels bits codificats amb l'algorisme de Huffman (*Huffman code bits*) amb l'ajuda de l'informació emmagatzemada en la part anterior (*Side Information*). Posteriorment s'ha de dur a terme la decodificació dels bits codificats amb l'algorisme de Huffman per, finalment, procedir a la requantificació dels mateixos amb l'ajuda dels *scalefactors* extrets.

Per a realitzar aquestes tres fases, s'ha utilitzat tres classes ben diferenciades que engloben totes les funcionalitat de cadascuna de les fases remarcades. Aquestes tres classes són: *FrameHeader, SideInfo* i *MainData*, respectivament. Dins de cada una de les classes es duen a terme petites rutines de tractament de l'informació de l'arxiu MP3 per a cada una de les fases.

En les parts que formen els dos objectius d'aquest Treball Fi de Carrera: interpretació i extracció de les components freqüencials de les trames i modificació de la guanyança global de cada trama donat un valor definit (definits més extensament en el punt *2.1. Objectius del projecte*), es segueixen camins diferents per arribar al seu objectiu final.

En la part de modificació de la guanyança global s'utilitzen principalment dos de les classes anteriorment descrites, *FrameHeader i SideInfo*, ja que no ens cal arribar al nivell de components freqüencials per a poder realitzar-la. De totes maneres, s'ha optat per a que en el codi s'arribi sempre a la interpretació de les components freqüencials, encara que s'estigui duen a terme una modificació de la guanyança global. Aquesta ha estat una decisió pròpia per a que sempre es segueixi el camí lògic de l'interpretació d'un arxiu MP3, necessari per a l'objectiu global del TFC (*Normalització d'un volum d'arxius en format MP3*), encara que es vulgui realitzar una modificació de la guanyança global de les trames. Així es dóna una visió més compacte de la implementació de les parts objectiu.

La idea de tota l'implementació d'aquest projecte, és formar part d'un entorn integrat on es dugui a terme la normalització completa d'arxius. La complexa realització d'aquesta normalització completa d'arxius obliga a encarar-la de manera particionada en dos mòduls diferents que integraràn l'entorn integrat on es dugui a terme tot el procés de normalització. Per això, el codi s'ha adequat per a la hipotètica futura integració en aquest entorn descrit però, a la vegada, s'ha implementat una execució autònoma que pugui fer veure les possibilitats del programari generat en aquest Treball Fi de Carrera.

Concretament, s'ha generat un producte final que demostra la correcta realització de l'objectiu de modificació de la guanyança global. Aquest agafa uns arguments entrats per l'usuari: arxiu MP3 a tracta i valors de guanyança per

cadascún dels dos canals d'un arxiu, *estèreo*, o d'un canal, *mono*. Aquests valors de guanyança entrats per l'usuari correspondrien, en l'entorn integrat, al factor de normalització generat en la part d'anàlisi de components freqüencials de les trames d'un volum d'arxius MP3 (*definida en el punt* ***2.1. Objectius del projecte***) realitza aquesta modificació de la guanyança.

En la part, que també forma part de l'objectiu d'aquest Treball Fi de Carrera, el tractament de les trames dels arxius MP3 per a dur a terme l'extracció de components freqüencials, degut a l'obtenció d'unes dades poc visuals que es realitza, s'ha optat per no generar un producte final. Per a poder comprobar el treball realitzat, s'ha inclòs en el codi tot un seguit d'explicacions i comentaris que ajuden a comprendre pas a pas tot el procés que es realitza en aquesta part. De la mateixa manera, també es descriu tot aquest procés en la present memòria.

5. Conclusions

5.1. Conclusió

En la confecció d'aquest projecte he hagut d'aprofundir molt en l'estructura dels arxius MP3, cosa que ha fet augmentar moltíssim els meus coneixements en aquest camp. Camp, el qual, està actualment de "rabiosa" actualitat i conegut per tothom. Per tant, ha estat un treball totalment útil i vertaderament aprofitable per a un possible futur ampli aprofundiment i dedicació en el camp del tractament de formats de compressió d'àudio.

A la vegada, el producte que es pot generar a partir d'aquest Treball Fi de Carrera, té una utilitat posterior eminentment pràctica donada l'extesa utilització actual d'aquests arxius, principalment a través de la xarxa d'Internet. D'aquesta manera, en la realització d'aquest projecte he intentat assolir un nivell de d'implementació totalment operatiu per a satisfer les necessitats pràctiques que pot solucionar.

5.2. Línies de futur

Com s'ha comentat anteriorment, la idea principal de la partició de l'objectiu global d'aquest projecte en dues parts ben diferenciades (per una banda, l'extracció i interpretació de les components freqüencials juntament amb la modificació de la guanyança global; i per una altra banda, l'anàlisi i comparació dels nivells de potència de les trames, extrets en la primera part, per a poder calcular un factor de normalització que s'aplicarà posteriorment, en la primera part), és la posterior integració dins d'una interfície, entorn, on es gestionin conjuntament totes les parts i doni un servei totalment útil per als usuaris que necessitin utilitzar la normalització d'un volum d'arxius en format MP3.

De la mateixa manera, dins de la part que s'ocupa aquest Treball Fi de Carrera s'ha assolit els objectius requerits però que, en una posterior ampliació es podria millorar en la seva implementació i camp d'acció.

En la interpretació de les capçaleres i, conseqüentment dels arxiu complets, s'ha optat per només descodificar els formats MPEG 1 i MPEG 2, deixant, doncs, de banda el format MPEG 2.5, degut a la poca utilitat que té en el camp dels arxius d'àudio, ja que utilitza unes freqüencies tan baixes que no es fan servir en la codificació d'arxius de música. Per tant, aquesta podria ser una possible línia de

futur: englobar els tres formats, ampliant l'actual, que només engloba els dos primers formats.

Altres petites ampliacions en la implementació podrien ser la possiblitat d'admetre arxius (trames) en *free format*, el qual és un format en el que no s'indiquen les característiques de la trama en la capçalera, com és el bitrate; que en l'actual implementació no s'ha admès degut a la seva poca utilització i la superior dificultat d'implementació que representa.

Una altra petita ampliació seria la implementació d'una rutina de comprobació del checksum, en els casos en els que s'utilitzi dins de les trames (camp *Protection bit* del *FrameHeader* igual a 0). Rutina que no s'ha implementat per a la poca utilització que té en els arxius MP3 i la poca necessitat que hi ha en la seva aplicació, ja que els arxius rares vegades incorporen el checksum i en les que el porten poques vegades hi han errors. Però com a utilitat de la que disposa el format, seria una possible ampliació de les funcionalitat que engloba l'actual projecte.

6. Bibliografia

6.1. Bibliografia digital

Aprofitant els avantatges que ens proporciona avui en dia la "xarxa de xarxes" per a poder recollir molta informació, la majoria de documentació per a la realització d'aquest projecte ha estat extreta d'Internet en dues fonts diferents: pàgines web i revistes electròniques:

- Web del *MP3'Tech*, http://www.mp3-tech.org
- Web del *Fraunhoffer IIS* dedicada al format MP3, http://www.iis.fhg.de/amm/techinf/layer3
- Web de recursos MPEG, http://www.mpeg.org
- Web del *Audio Engineering Society Convention*, http://www.aes.org
- Revista electrònica *IEEE Signal Processing Magazine*, article: *MPEG Digital Audio Coding*, Setembre del 1997.
- Revista electrònica *Digital Technical Journal*, Vol. 5 no. 2, article: *Digital Audio Compression*, autor Davis Yen Pan, Primavera del 1993. http://www.iocon.com/resource/docs/pdf/Digital_Audio_Compression_01oct1993DTJA03P8.pdf
- PDF: *A tutorial on MPEG/Audio Compression*, autor Davis Pan, http://www.iocon.com/resource/docs/pdf/mpegaud.pdf
- PDF: *Introduction to MPEG Layer III*, http://www.ece.cmu.edu/~ee545/MP3/docs/MP3_lecture.pdf

6.2. Documentació

La documentació en "paper", encara que aconseguida també a partir d'Internet engloba únicament el recull dels estàndards de compressió d'àudio oficials de l'organització d'estàndards internacional (ISO). Aquests són:

- ISO/IEC, ISO/IEC 13818-3: Information Technology -Generic Coding of Moving Pictures and Associated Audio (Audio Part).
- ISO/IEC, ISO/IEC 11172-3: Coding of moving pictures associated audio for digital storage media at up to about 1.5 Mbit/s (Part 3 Audio).

7. Annex

7.1. Procediment principal

```
/* Definició del procediment principal que llançarà tota
   l'execució de l'aplicació.                          */

#include <stdlib.h>

#include "defs.h"
#include "FrameHeader.h"
#include "SideInfo.h"
#include "MainData.h"
#include "MP3Info.h"

int main(int argc, char *argv[])
{
  BYTE opcio;     // Descriu l'acció que volem realitzar. Sempre es realitzarà
                  // l'anàlisi de componets freqüencials però si a més opcio==1
                  // llavors pujarem/baixarem la guanyança dels frames en
                  // relació al valor de la variable gain.
  float gain[2];  // Valors d'amplificació/atenuació de la potència per a cada
                  // cada canal per separat per tots els frames.

  switch(argc)
  {
    case 2:
      /* Només farem l'extracció de components frecuencials */
      opcio=0;
      break;
    case 4:
      /* A part de l'extracció de components, modificarem la guanyança
         global de les trames a partir d'un valor entrat per l'usuari.
         Un valor de guanyança per cada canal.                        */
      opcio=1;
      gain[0] = atof(argv[2]);  // Valor de guanyança pel canal dret
      printf("\n\nFixada una guanyanca pel canal dret de:       %3.2f", gain[0]);
      gain[1] = atof(argv[3]);  // Valor de guanyança pel canal esquerre
      printf("\nFixada una guanyanca pel canal esquerre de:   %3.2f\n", gain[1]);
      break;
    default:
      printf("\nError en els arguments!");
      exit(0);
  }

  /* Arranquem tot el procés cridant a la classe MP3Info, que obrirà
     l'arxiu MP3 passat com a argument                              */
  MP3Info infoMP3(argv[1]);

  buf_pointer[0]=0;   // Posem el punter al principi ja que el buffer està buit
```

```
 infoMP3.readFrame();       // Recollida dels paràmetres de cada frame

 while (!infoMP3.final())
 {
   if (opcio == 1)
     /* Modifiquem el volum del frame segons el valor gain */
     infoMP3.setGain(gain[0], gain[1]);

   infoMP3.readFrame();
 }

 return 0;
}
```

7.2. Classe MP3Info (*MP3Info.h*)

```
/* Definició de la classe que emmagatzemarà tota la informació
   de l'arxiu MP3 que estem tractant.                          */

#include <math.h>


class MP3Info {
  private:
    FILE *fp;
    BOOL endOfFile;
    DBYTE FrameSize;
    long FrameSizeTotal;
    DBYTE gain[2][2];
    BOOL is_mono;      // Ens dirà si el frame té només un canal
    BYTE mpegID;       // Ens dirà si es tracta de MPEG 1 o 2.
    BYTE crc_check;    // Ens dirà si el frame fa una comprobació del CRC
    DBYTE nFrame;      // Indica el número de frame actual

    FrameInformation FrameInfo;


  public:
    MP3Info();
    MP3Info(const char *mp3_filename);  // builds the object and open the file
    void open(const char *mp3_filename);
    void close();
    ~MP3Info();
    void getInfo();
    void readFrame();
    void readFile();
    DBYTE readHeader();
    void setFrameSize(BYTE i, BYTE b, BYTE s, BYTE p);
    void sortBuffers();
    BOOL final();
    void setGain(float right_gain, float left_gain);
    void writeGain();
    void writeGainMpeg1();
```

```
    void writeGainMpeg2();
    void setFrameInfo(BYTE nCh, BYTE brate, BYTE i);

    int read_freq(FrameInformation *frame_information);

};

MP3Info::MP3Info()
{
}

MP3Info::MP3Info(const char *mp3_filename)
{
  endOfFile=0;
  buf_actius=0;
  nFrame=0;
  FrameSizeTotal=0;

  if ((fp=fopen(mp3_filename, "r+b"))==NULL)
  {
    exit(1);     // Error en l'obertura del fitxer MP3
  }

  printf("\nTractant frame:        ");
}

MP3Info::~MP3Info()
{
  fclose(fp);
}

void MP3Info::open(const char *mp3_filename)
{
  if ((fp=fopen(mp3_filename, "r+b"))==NULL)
  {
    exit(1);     // Error en l'obertura del fitxer MP3
  }

  printf("\nTractant frame:        ");
}

void MP3Info::close()
{
  fclose(fp);
}

void MP3Info::getInfo()
{
}

void MP3Info::readFrame()
{
  BYTE granule, channel, nGranules, nChannels;
  DBYTE nBeforeHeader;

  buf_actius++;
  nFrame++;

  printf("\b\b\b\b\b\b");
```

```
printf("%6ld", nFrame);

/* Fem una reordenació dels buffers per tal de posar
   l'actual frame al primer buffer.                */
sortBuffers();

buf_pointer[0]=0;

/* No agafarem tot el frame si no només el header, ja
   que si no no podem saber la mida del frame.        */
nBeforeHeader = readHeader();

/* Sumem el nombre de bytes que hi ha anteriors al header
   al número total de bytes de frames per a poder saber
   on estàn les posicions absolutes de tots els frames.    */
FrameSizeTotal = FrameSizeTotal + nBeforeHeader;

/* Omplim els camps del header del frame actual */
FrameHeader fh;

/* A partir dels paràmetres del header calculem la mida del frame */
setFrameSize(fh.getID(), fh.getBitrate(), fh.getSampling(), fh.getPadding());

/* Llegim la part del frame que ens queda i el posem al buffer */
readFile();
if (endOfFile)
{
     printf("\b\b\b\b\b\b");
     printf("%6ld", nFrame-1);
     printf("\n\nFinalitzat tot el proces! \n");
     return;
}

/* Omplim els camps del Side Info */
SideInfo si(fh.getMode(), fh.getID());

/* Agafem els valors de protection, gain, mode i ID per si hem de fer
   una modificació en el volum dels frames (opcio==1).     */
if (fh.getProtection()==0) // En aquest cas hi haurà comprobació del CRC
   crc_check=1;
else                       // En aquest cas no hi haurà comprobació del CRC
   crc_check=0;

if (fh.getMode()==3)
{
   is_mono=1;
   nChannels=1;
}
else
{
   is_mono=0;
   nChannels=2;
}

if ((mpegID=fh.getID()) == 1)
   nGranules=2;
else
   nGranules=1;
```

```
  for (granule=0 ; granule<nGranules ; granule++)
  {
    for (channel=0 ; channel<nChannels ; channel++)
      gain[granule][channel] = si.getGlobalGain(granule, channel);
  }

  /* Tot seguit agafem ja el main_data */
  MainData md(***(FrameInfo.values), si.getSideInfoData(), fh.getMode(), fh.getModeExt(), fh.getID());

  /* Omplir les dades del frame, posant-les a FrameInformation.  */
  setFrameInfo(nChannels, fh.getBitrate(), fh.getID());
}

void MP3Info::readFile()
/* Aquesta funció ens llegirà el frame de l'arxiu MP3 i el posarà dins
   del buffer. Si el buffer ja tenia emmagatzemat un frame anterior, afegirà
   el nou frame al final de l'anterior fins a omplir el buffer senzer, o
   sigui 515 bytes.                                          */
{
  DBYTE i;
  DBYTE fs;
  int c;

  fs = FrameSize - buf_pointer[0];

  for (i=0 ; i<fs && !endOfFile ; i++) // Anem agafant byte a byte de l'arxiu
  {
    c=getc(fp);

    if (c==EOF)
      endOfFile=1;
    else
    {
      buffer[0][buf_pointer[0]]=c;
      buf_pointer[0]++;
    }
  }
}

DBYTE MP3Info::readHeader()
/* Busca i recull tot el header del frame. Retorna els bytes sobrants
   anteriors al principi del frame, o sigui, anteriors al syncword.    */
{
  BYTE temp, i;
  BYTE s=0;
  BOOL trobat=0;
  DBYTE nBytes=0;
  int c;

  c=getc(fp);
  nBytes++;

  while (!trobat)  // Anem agafant byte a byte de l'arxiu fins trobar syncword
  {
    for(i=0;i<8 && !trobat;i++)
    {
      /* Comencem a extreure bit a bit del byte que hem llegit */
      temp=c;
      temp=temp<<i;
```

```
      temp=temp>>7;

      /* Mirem si aquest bit pot formar part del syncword o no */
      if (temp==1)
        s++;
      else if (temp==0)  // Ens ve un 0, no pot ser el syncword. Si era
                         // precedit per 1's, no formaven part del syncword.
        s=0;


      /* Mirem si ja tenim el syncword completat */
      if (s==12)
        trobat=1;
    }
    if(trobat)         // Hem trobat el syncword
    {
      buffer[0][buf_pointer[0]]=c;
      buf_pointer[0]++;

      /* Com que és el primer syncword, recollim el header.  */
      for(i=0; i<3 ; i++, buf_pointer[0]++)
      {
        c=getc(fp);
        nBytes++;
        buffer[0][buf_pointer[0]]=c;
      }

      s=0;
    }
    else if(!trobat)
    {
      if (s>0)     // Els bits d'aquest byte poden formar part del syncword
      {
        if (s>8)
        {
          buffer[0][buf_pointer[0]]=c;
          buf_pointer[0]++;
        }
        else      // Els bits d'aquest byte poden formar part del syncword
                  // però no els de l'anterior byte llegit.
        {
          buffer[0][0]=c;
          buf_pointer[0]=1;
        }
      }

      c=getc(fp);
      nBytes++;
    }
  }
  return (nBytes-buf_pointer[0]);
}

void MP3Info::setFrameSize(BYTE i, BYTE b, BYTE s, BYTE p)
{
  switch(b)
  {
    case 1:
      if (i==1)   // Es tracta de MPEG 1
```

```
      FrameSize=144*32000;
    else       // Es tracta de MPEG 2
      FrameSize=144*8000;
    break;
  case 2:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*40000;
    else       // Es tracta de MPEG 2
      FrameSize=144*16000;
    break;
  case 3:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*48000;
    else       // Es tracta de MPEG 2
      FrameSize=144*24000;
    break;
  case 4:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*56000;
    else       // Es tracta de MPEG 2
      FrameSize=144*32000;
    break;
  case 5:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*64000;
    else       // Es tracta de MPEG 2
      FrameSize=144*40000;
    break;
  case 6:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*80000;
    else       // Es tracta de MPEG 2
      FrameSize=144*48000;
    break;
  case 7:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*96000;
    else       // Es tracta de MPEG 2
      FrameSize=144*56000;
    break;
  case 8:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*112000;
    else       // Es tracta de MPEG 2
      FrameSize=144*64000;
    break;
  case 9:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*128000;
    else       // Es tracta de MPEG 2
      FrameSize=144*80000;
    break;
  case 10:
    if (i==1)   // Es tracta de MPEG 1
      FrameSize=144*160000;
    else       // Es tracta de MPEG 2
      FrameSize=144*96000;
    break;
  case 11:
```

```
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=144*192000;
        else        // Es tracta de MPEG 2
          FrameSize=144*112000;
        break;
      case 12:
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=144*224000;
        else        // Es tracta de MPEG 2
          FrameSize=144*128000;
        break;
      case 13:
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=144*256000;
        else        // Es tracta de MPEG 2
          FrameSize=144*144000;
        break;
      case 14:
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=144*320000;
        else        // Es tracta de MPEG 2
          FrameSize=144*160000;
        break;
    }

    switch(s)
    {
      case 0:
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=FrameSize/44100;
        else
          FrameSize=FrameSize/22050;
        break;
      case 1:
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=FrameSize/48000;
        else
          FrameSize=FrameSize/24000;
        break;
      case 2:
        if (i==1)   // Es tracta de MPEG 1
          FrameSize=FrameSize/32000;
        else
          FrameSize=FrameSize/16000;
        break;
    }

    FrameSize = FrameSize + p;
    FrameSizeTotal = FrameSizeTotal + FrameSize;
}

void MP3Info::sortBuffers()
{
    BYTE i;
    DBYTE j;

    if (buf_actius>1 && buf_actius<10)    // Encara no tenim tots els buffers plens
    {
      for (i=buf_actius-2 ; i>0 ; i--)  // Correm una posició endavant els buffers
```

```
    {
      for (j=0 ; j<=buf_pointer[i] ; j++)
        buffer[i+1][j]=buffer[i][j];

      /* Actualitzem també els punters */
      buf_pointer[i+1]=buf_pointer[i];
      buf_actual[i+1]=buf_actual[i];
      buf_restant[i+1]=buf_restant[i];
    }

    for (j=0 ; j<=buf_pointer[i] ; j++)
      buffer[i+1][j]=buffer[i][j];

    /* Actualitzem també els punters */
    buf_pointer[i+1]=buf_pointer[i];
    buf_actual[i+1]=buf_actual[i];
    buf_restant[i+1]=buf_restant[i];
  }
  else if (buf_actius==10)            // Tenim tots els buffers plens
  {
    buf_actius=9;

    for (i=7 ; i>0 ; i--)  // Correm una posició endavant els buffers
                           // i l'últim buffer desapareix
    {
      for (j=0 ; j<=buf_pointer[i] ; j++)
        buffer[i+1][j]=buffer[i][j];

      /* Actualitzem també els punters */
      buf_pointer[i+1]=buf_pointer[i];
      buf_actual[i+1]=buf_actual[i];
      buf_restant[i+1]=buf_restant[i];
    }

    for (j=0 ; j<=buf_pointer[i] ; j++)
      buffer[i+1][j]=buffer[i][j];

    /* Actualitzem també els punters */
    buf_pointer[i+1]=buf_pointer[i];
    buf_actual[i+1]=buf_actual[i];
    buf_restant[i+1]=buf_restant[i];
  }
}

BOOL MP3Info::final()
{
  return (endOfFile);
}

void MP3Info::setGain(float right_gain, float left_gain)
{
  BYTE granule, channel, nChannels, nGranules;
  float amplify[2];

  /* Amplifiquem/atenuem (sumem o restem) al global_gain segons
    el valor logarítmic de la g.                          */
  if (is_mono)
    nChannels=1;
  else
```

```
    nChannels=2;

  if (mpegID == 1)    // Es MPEG 1
    nGranules=2;
  else                // Es MPEG 2
    nGranules=1;

  /* Passem primer el valor d'atenuació/amplificació a representació
    logarítmica per a aplicar la suma posterior a la guanyança global
    de cada frame.                                          */
  amplify[0] = 20*(log10((double)right_gain));
  amplify[1] = 20*(log10((double)left_gain));

  /* Amplifiquem/atenuem la senyal amb la suma (positiva/negativa) de g */
  for (granule=0 ; granule<nGranules ; granule++)
  {
    for (channel=0 ; channel<nChannels ; channel++)
      gain[granule][channel] = gain[granule][channel] + (DBYTE)amplify[channel];
  }

  /* Ara ja podem escriure el global_gain modificat */
  writeGain();
}

void MP3Info::writeGain()
{
  /* Escrivim els nous valors de gain a l'arxiu MP3 */
  if (mpegID == 1)    // Es MPEG 1
    writeGainMpeg1();
  else                // Es MPEG 2
    writeGainMpeg2();

  /* Resituem el punter de l'arxiu per deixar-lo preparat per llegir
    el proper frame.                                        */
  if (fseek(fp, FrameSizeTotal, SEEK_SET)) // Ens hi situem
  {
    printf("\nHi ha hagut un error en l'arxiu!");
    exit(1);
  }
}

void MP3Info::writeGainMpeg1()
{
  BYTE temp1, temp2;
  long desp;
  DBYTE c[2];

  if (is_mono)
  {
    /* Les posicions relatives que ocupen les variables global_gain dins del
      frame sempre seràn les mateixes, en el cas de tenir un canal:
        - global_gain[0][0] va des de l'últim bit del byte 8 fins al
          penúltim bit del byte 9 (sempre partint del principi de frame).
        - global_gain[1][0] va des del tercer bit del byte 16 fins al
          segon bit del byte 17 (sempre partint del principi de frame).
      En totes aquestes posicions hi haurem de sumar dos bytes en el cas
      de que hi hagi una comprobació del CRC en el frame (crc_check==1).
      Per tant, la posició absoluta l'obtindrem restant del FrameSizeTotal
      la llargada del frame actual (FrameSize) per anar directament al
```

```
  principi d'aquest frame i sumant-li la posició de global_gain descrita
  anteriorment.                                          */

/* Anem pel global_gain[0][0] */
desp = (FrameSizeTotal-FrameSize)+8;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Llegim els dos bytes que estaràn implicats */
c[0]=getc(fp);
c[1]=getc(fp);

/* Modifiquem els dos bytes amb el nou valor de gain */
temp1 = c[0];
temp1 = temp1 >> 1;
temp1 = temp1 << 1;
temp2 = gain[0][0];
temp2 = temp2 >> 7;
c[0] = temp1 | temp2;
temp1 = c[1];
temp1 = temp1 << 7;
temp1 = temp1 >> 7;
temp2 = gain[0][0];
temp2 = temp2 << 1;
c[1] = temp1 | temp2;

/* Ja podem escriure el valor a l'arxiu */
desp = (FrameSizeTotal-FrameSize)+8;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Escrivim els dos bytes */
putc(c[0], fp);
putc(c[1], fp);

/* Anem pel global_gain[1][0] */
desp = (FrameSizeTotal-FrameSize)+16;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Llegim els dos bytes que estaràn implicats */
c[0]=getc(fp);
c[1]=getc(fp);

/* Modifiquem els dos bytes amb el nou valor de gain */
temp1 = c[0];
temp1 = temp1 >> 6;
```

```
  temp1 = temp1 << 6;
  temp2 = gain[1][0];
  temp2 = temp2 >> 2;
  c[0] = temp1 | temp2;
  temp1 = c[1];
  temp1 = temp1 << 2;
  temp1 = temp1 >> 2;
  temp2 = gain[1][0];
  temp2 = temp2 << 6;
  c[1] = temp1 | temp2;

  /* Ja podem escriure el valor a l'arxiu */
  desp = (FrameSizeTotal-FrameSize)+16;
  if (crc_check)
    desp = desp + 2;
  if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
  {
    printf("\nHi ha hagut un error en l'arxiu!");
    exit(1);
  }
  /* Escrivim els dos bytes */
  putc(c[0], fp);
  putc(c[1], fp);
}
else
{
  /* Les posicions relatives que ocupen les variables global_gain dins del
     frame sempre seràn les mateixes, en el cas de tenir dos canals:
       - global_gain[0][0] va des del segon bit del byte 9 fins al
         primer bit del byte 10 (sempre partint del principi de frame).
       - global_gain[0][1] va des del cinquè bit del byte 16 fins al
         quart bit del byte 17 (sempre partint del principi de frame).
       - global_gain[1][0] va des de l'últim bit del byte 23 fins al
         penúltim bit del byte 24 (sempre partint del principi de frame).
       - global_gain[1][1] va des del tercer bit del byte 31 fins al
         segon bit del byte 32 (sempre partint del principi de frame).
     En totes aquestes posicions hi haurem de sumar dos bytes en el cas
     de que hi hagi una comprobació del CRC en el frame (crc_check==1).
     Per tant, la posició absoluta l'obtindrem restant del FrameSizeTotal
     la llargada del frame actual (FrameSize) per anar directament al
     principi d'aquest frame i sumant-li la posició de global_gain descrita
     anteriorment.                                          */

  /* Anem pel global_gain[0][0] */
  desp = (FrameSizeTotal-FrameSize)+9;
  if (crc_check)
    desp = desp + 2;
  if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
  {
    printf("\nHi ha hagut un error en l'arxiu!");
    exit(1);
  }
  /* Llegim els dos bytes que estaràn implicats */
  c[0]=getc(fp);
  c[1]=getc(fp);

  /* Modifiquem els dos bytes amb el nou valor de gain */
  temp1 = c[0];
  temp1 = temp1 >> 7;
```

```
temp1 = temp1 << 7;
temp2 = gain[0][0];
temp2 = temp2 >> 1;
c[0] = temp1 | temp2;
temp1 = c[1];
temp1 = temp1 << 1;
temp1 = temp1 >> 1;
temp2 = gain[0][0];
temp2 = temp2 << 7;
c[1] = temp1 | temp2;

/* Ja podem escriure el valor a l'arxiu */
desp = (FrameSizeTotal-FrameSize)+9;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Escrivim els dos bytes */
putc(c[0], fp);
putc(c[1], fp);

/* Anem pel global_gain[0][1] */
desp = (FrameSizeTotal-FrameSize)+16;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Llegim els dos bytes que estaràn implicats */
c[0]=getc(fp);
c[1]=getc(fp);

/* Modifiquem els dos bytes amb el nou valor de gain */
temp1 = c[0];
temp1 = temp1 >> 4;
temp1 = temp1 << 4;
temp2 = gain[0][1];
temp2 = temp2 >> 4;
c[0] = temp1 | temp2;
temp1 = c[1];
temp1 = temp1 << 4;
temp1 = temp1 >> 4;
temp2 = gain[0][1];
temp2 = temp2 << 4;
c[1] = temp1 | temp2;

/* Ja podem escriure el valor a l'arxiu */
desp = (FrameSizeTotal-FrameSize)+16;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
```

```
}
/* Escrivim els dos bytes */
putc(c[0], fp);
putc(c[1], fp);

/* Anem pel global_gain[1][0] */
desp = (FrameSizeTotal-FrameSize)+23;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Llegim els dos bytes que estaràn implicats */
c[0]=getc(fp);
c[1]=getc(fp);

/* Modifiquem els dos bytes amb el nou valor de gain */
temp1 = c[0];
temp1 = temp1 >> 1;
temp1 = temp1 << 1;
temp2 = gain[1][0];
temp2 = temp2 >> 7;
c[0] = temp1 | temp2;
temp1 = c[1];
temp1 = temp1 << 7;
temp1 = temp1 >> 7;
temp2 = gain[1][0];
temp2 = temp2 << 1;
c[1] = temp1 | temp2;

/* Ja podem escriure el valor a l'arxiu */
desp = (FrameSizeTotal-FrameSize)+23;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Escrivim els dos bytes */
putc(c[0], fp);
putc(c[1], fp);

/* Anem pel global_gain[1][1] */
desp = (FrameSizeTotal-FrameSize)+31;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Llegim els dos bytes que estaràn implicats */
c[0]=getc(fp);
c[1]=getc(fp);

/* Modifiquem els dos bytes amb el nou valor de gain */
```

```
    temp1 = c[0];
    temp1 = temp1 >> 6;
    temp1 = temp1 << 6;
    temp2 = gain[1][1];
    temp2 = temp2 >> 2;
    c[0] = temp1 | temp2;
    temp1 = c[1];
    temp1 = temp1 << 2;
    temp1 = temp1 >> 2;
    temp2 = gain[1][1];
    temp2 = temp2 << 6;
    c[1] = temp1 | temp2;

    /* Ja podem escriure el valor a l'arxiu */
    desp = (FrameSizeTotal-FrameSize)+31;
    if (crc_check)
      desp = desp + 2;
    if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
    {
      printf("\nHi ha hagut un error en l'arxiu!");
      exit(1);
    }
    /* Escrivim els dos bytes */
    putc(c[0], fp);
    putc(c[1], fp);
  }
}

void MP3Info::writeGainMpeg2()
{
  BYTE temp1, temp2;
  long desp;
  DBYTE c[2];

  if (is_mono)
  {
    /* La posició relativa que ocupa la variable global_gain dins del
       frame sempre serà la mateixa, en el cas de tenir un canal:
         - global_gain[0][0] va des del setè bit del byte 7 fins al
           sisè bit del byte 8 (sempre partint del principi de frame).
       Hi haurem de sumar dos bytes en el cas de que hi hagi una comprobació
       del CRC en el frame (crc_check==1).
       Per tant, la posició absoluta l'obtindrem restant del FrameSizeTotal
       la llargada del frame actual (FrameSize) per anar directament al
       principi d'aquest frame i sumant-li la posició de global_gain descrita
       anteriorment.                                          */

    /* Anem pel global_gain[0][0] */
    desp = (FrameSizeTotal-FrameSize)+7;
    if (crc_check)
      desp = desp + 2;
    if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
    {
      printf("\nHi ha hagut un error en l'arxiu!");
      exit(1);
    }
    /* Llegim els dos bytes que estaràn implicats */
    c[0]=getc(fp);
    c[1]=getc(fp);
```

```
  /* Modifiquem els dos bytes amb el nou valor de gain */
  temp1 = c[0];
  temp1 = temp1 >> 2;
  temp1 = temp1 << 2;
  temp2 = gain[0][0];
  temp2 = temp2 >> 6;
  c[0] = temp1 | temp2;
  temp1 = c[1];
  temp1 = temp1 << 6;
  temp1 = temp1 >> 6;
  temp2 = gain[0][0];
  temp2 = temp2 << 2;
  c[1] = temp1 | temp2;

  /* Ja podem escriure el valor a l'arxiu */
  desp = (FrameSizeTotal-FrameSize)+7;
  if (crc_check)
    desp = desp + 2;
  if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
  {
    printf("\nHi ha hagut un error en l'arxiu!");
    exit(1);
  }
  /* Escrivim els dos bytes */
  putc(c[0], fp);
  putc(c[1], fp);
}
else
{
  /* Les posicions relatives que ocupen les variables global_gain dins del
     frame sempre seràn les mateixes, en el cas de tenir dos canals:
       - global_gain[0][0] va des de l'últim bit del byte 7 fins al
         penúltim bit del byte 8 (sempre partint del principi de frame).
       - global_gain[0][1] va des del penúltim bit del byte 15 fins al
         sisè bit del byte 16 (sempre partint del principi de frame).
     Hi haurem de sumar dos bytes en el cas de que hi hagi una comprobació
     del CRC en el frame (crc_check==1).
     Per tant, la posició absoluta l'obtindrem restant del FrameSizeTotal
     la llargada del frame actual (FrameSize) per anar directament al
     principi d'aquest frame i sumant-li la posició de global_gain descrita
     anteriorment.                                          */

  /* Anem pel global_gain[0][0] */
  desp = (FrameSizeTotal-FrameSize)+7;
  if (crc_check)
    desp = desp + 2;
  if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
  {
    printf("\nHi ha hagut un error en l'arxiu!");
    exit(1);
  }
  /* Llegim els dos bytes que estaràn implicats */
  c[0]=getc(fp);
  c[1]=getc(fp);

  /* Modifiquem els dos bytes amb el nou valor de gain */
  temp1 = c[0];
  temp1 = temp1 >> 1;
```

```
temp1 = temp1 << 1;
temp2 = gain[0][0];
temp2 = temp2 >> 7;
c[0] = temp1 | temp2;
temp1 = c[1];
temp1 = temp1 << 7;
temp1 = temp1 >> 7;
temp2 = gain[0][0];
temp2 = temp2 << 1;
c[1] = temp1 | temp2;

/* Ja podem escriure el valor a l'arxiu */
desp = (FrameSizeTotal-FrameSize)+7;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Escrivim els dos bytes */
putc(c[0], fp);
putc(c[1], fp);

/* Anem pel global_gain[0][1] */
desp = (FrameSizeTotal-FrameSize)+15;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
}
/* Llegim els dos bytes que estaràn implicats */
c[0]=getc(fp);
c[1]=getc(fp);

/* Modifiquem els dos bytes amb el nou valor de gain */
temp1 = c[0];
temp1 = temp1 >> 2;
temp1 = temp1 << 2;
temp2 = gain[0][1];
temp2 = temp2 >> 6;
c[0] = temp1 | temp2;
temp1 = c[1];
temp1 = temp1 << 6;
temp1 = temp1 >> 6;
temp2 = gain[0][1];
temp2 = temp2 << 2;
c[1] = temp1 | temp2;

/* Ja podem escriure el valor a l'arxiu */
desp = (FrameSizeTotal-FrameSize)+15;
if (crc_check)
  desp = desp + 2;
if (fseek(fp, desp, SEEK_SET)) // Ens hi situem
{
  printf("\nHi ha hagut un error en l'arxiu!");
  exit(1);
```

```
    }
    /* Escrivim els dos bytes */
    putc(c[0], fp);
    putc(c[1], fp);
  }
}

void MP3Info::setFrameInfo(BYTE nCh, BYTE brate, BYTE i)
{
  DBYTE bitrate;

  /* Entrem número de canals */
  FrameInfo.n_channels = nCh;

  /* Agafem el bitrate */
  switch(brate)
  {
    case 1:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=32000;
      else       // Es tracta de MPEG 2
        bitrate=8000;
      break;
    case 2:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=40000;
      else       // Es tracta de MPEG 2
        bitrate=16000;
      break;
    case 3:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=48000;
      else       // Es tracta de MPEG 2
        bitrate=24000;
      break;
    case 4:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=56000;
      else       // Es tracta de MPEG 2
        bitrate=32000;
      break;
    case 5:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=64000;
      else       // Es tracta de MPEG 2
        bitrate=40000;
      break;
    case 6:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=80000;
      else       // Es tracta de MPEG 2
        bitrate=48000;
      break;
    case 7:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=96000;
      else       // Es tracta de MPEG 2
        bitrate=56000;
      break;
```

```
    case 8:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=112000;
      else        // Es tracta de MPEG 2
        bitrate=64000;
      break;
    case 9:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=128000;
      else        // Es tracta de MPEG 2
        bitrate=80000;
      break;
    case 10:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=160000;
      else        // Es tracta de MPEG 2
        bitrate=96000;
      break;
    case 11:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=192000;
      else        // Es tracta de MPEG 2
        bitrate=112000;
      break;
    case 12:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=224000;
      else        // Es tracta de MPEG 2
        bitrate=128000;
      break;
    case 13:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=256000;
      else        // Es tracta de MPEG 2
        bitrate=144000;
      break;
    case 14:
      if (i==1)   // Es tracta de MPEG 1
        bitrate=320000;
      else        // Es tracta de MPEG 2
        bitrate=160000;
      break;
  }

  /* Entrem la durada en mi·lisegons */
  FrameInfo.msec = (FrameSize/bitrate) * 1000.0;
}

int MP3Info::read_freq(FrameInformation *frame_information)  //returns 0 if ok; -1 if EOF ...
{
  frame_information = &FrameInfo;

  return (0);
}
```

7.3. Classe FrameHeader (*FrameHeader.h*)

```
/* Definició de la classe que durà a terme la lectura dels paràmetres
   del header que ens interessen de cada frame.                    */

class FrameHeader
{
  private:
    /* Camps del header de cada frame */
    BYTE ID;
    BYTE layer;
    BYTE protection;
    BYTE bitrate;
    BYTE sampling;
    BYTE padding;
    BYTE mode;
    BYTE mode_ext;

  public:
    FrameHeader();
    ~FrameHeader();
    void SearchSync();
    void setID();
    void setLayer();
    void setProtection();
    void setBitrate();
    void setSampling();
    void setPadding();
    void setMode();
    void setModeExt();
    BYTE getID();
    BYTE getLayer();
    BYTE getProtection();
    BYTE getBitrate();
    BYTE getSampling();
    BYTE getPadding();
    BYTE getMode();
    BYTE getModeExt();
    void GoToFinal();
    void runErrorCheck();
    void runBits(BYTE nBits);
};

FrameHeader::FrameHeader()
{
  // Ens situem al principi del buffer
  buf_actual[0]=0;

  // Busquem el syncword i el llegim
  SearchSync();

  // S'ha trobat syncword, anem a llegir els camps del header
  setID();
  setLayer();
  setProtection();
  setBitrate();
  setSampling();
```

```
  setPadding();
  runBits(1);          // Correspondria a Private_bit
  setMode();
  setModeExt();
  GoToFinal();
  runErrorCheck();     // Correspondria al Error_check
}

FrameHeader::~FrameHeader()
{
}

void FrameHeader::SearchSync()
/* Procediment que buscarà el patró de la paraula de sincronització (syncword). */
{
  BYTE s, i, temp;
  BOOL trobat=0;

  s=0;
  while (!trobat)  // Anem agafant byte a byte del buffer
  {
    for(i=0;i<8 && !trobat;i++)
    {
      /* Comencem a extreure bit a bit del byte que hem llegit */
      temp=buffer[0][buf_actual[0]];
      temp=temp<<i;
      temp=temp>>7;

      /* Mirem si aquest bit pot formar part del syncword o no */
      if (temp==1)
        s++;
      else if (temp==0 && s>0)  // Ens ve un 0 i era precedit per 1's que
        s=0;                    // no formaven part del syncword (<12).

      /* Mirem si ja tenim el syncword completat */
      if (s==12)
      {
        if ((i+1)<8)  // Queda bits per llegir dins del byte que llegíem
        {
          buf_restant[0]=i+1;
        }
        else          // No queden bits per llegir dins del byte actual
        {
          buf_restant[0]=0;
        }
        trobat=1;
      }
    }
    if (!trobat)
    {
      buf_actual[0]++;
    }
  }
}

void FrameHeader::setID()
/* Aquí obtindrem el valor del paràmetre ID, o sigui, l'algoritme que segons
  ISO serà:
    -ID=0, MPEG Version 2 (ISO/IEC 13818-3)
```

```
    -ID=1, MPEG Version 1 (ISO/IEC 11172-3)                    */
{
  BYTE temp, i;

  ID=0;

  for (i=0;i<1;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    ID=ID<<1;
    ID=ID | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setLayer()
/* Aquí obtindrem el valor del paràmetre layer, que segons ISO serà:
    -layer=0, reservat
    -layer=1, Layer III
    -layer=2, Layer II
    -layer=3, Layer I                                          */
{
  BYTE temp, i;

  layer=0;

  for (i=0;i<2;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    layer=layer<<1;
    layer=layer | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setProtection()
{
  BYTE temp, i;

  protection=0;

  for (i=0;i<1;i++)
  {
    if (buf_restant[0]==0)
    {
```

```
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    protection=protection<<1;
    protection=protection | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setBitrate()
/* Aquí obtindrem el valor del paràmetre bitrate, que segons ISO serà:
    -Si estem en l'algoritme MPEG 1 (ID=1):
       -bitrate=0, free
       -bitrate=1, 32 kbps
       -bitrate=2, 40 kbps
       -bitrate=3, 48 kbps
       -bitrate=4, 56 kbps
       -bitrate=5, 64 kbps
       -bitrate=6, 80 kbps
       -bitrate=7, 96 kbps
       -bitrate=8, 112 kbps
       -bitrate=9, 128 kbps
       -bitrate=10, 160 kbps
       -bitrate=11, 192 kbps
       -bitrate=12, 224 kbps
       -bitrate=13, 256 kbps
       -bitrate=14, 320 kbps

    -Si estem en l'algoritme MPEG 2 (ID=0):
       -bitrate=0, free
       -bitrate=1, 8 kbps
       -bitrate=2, 16 kbps
       -bitrate=3, 24 kbps
       -bitrate=4, 32 kbps
       -bitrate=5, 40 kbps
       -bitrate=6, 48 kbps
       -bitrate=7, 56 kbps
       -bitrate=8, 64 kbps
       -bitrate=9, 80 kbps
       -bitrate=10, 96 kbps
       -bitrate=11, 112 kbps
       -bitrate=12, 128 kbps
       -bitrate=13, 144 kbps
       -bitrate=14, 160 kbps                              */
{
  BYTE temp, i;

  bitrate=0;

  for (i=0;i<4;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
```

```
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    bitrate=bitrate<<1;
    bitrate=bitrate | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setSampling()
/* Aquí obtindrem el valor del paràmetre sampling, que segons ISO serà:
    -Si estem en l'algoritme MPEG 1 (ID=1):
       -sampling=0, 44.1 KHz
       -sampling=1, 48 KHz
       -sampling=2, 32 KHz
       -sampling=3, reservat

    -Si estem en l'algoritme MPEG 2 (ID=0):
       -sampling=0, 22.05 KHz
       -sampling=1, 24 KHz
       -sampling=2, 16 KHz
       -sampling=3, reservat                              */
{
  BYTE temp, i;

  sampling=0;

  for (i=0;i<2;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    sampling=sampling<<1;
    sampling=sampling | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setPadding()
{
  BYTE temp, i;

  padding=0;

  for (i=0;i<1;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
```

```
    padding=padding<<1;
    padding=padding | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setMode()
/* Aquí obtindrem el valor del paràmetre mode, que segons ISO serà:
    -mode=0, Stereo
    -mode=1, Joint Stereo
    -mode=2, Dual Channel
    -mode=3, Single Channel                                    */
{
  BYTE temp, i;

  mode=0;

  for (i=0;i<2;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    mode=mode<<1;
    mode=mode | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::setModeExt()
/* Aquí obtindrem el valor del paràmetre mode_ext, usats en el mode
  joint_stereo, que segons ISO serà:
    -mode_ext=0, Intensity_stereo=off, Ms_stereo=off
    -mode_ext=1, Intensity_stereo=on, Ms_stereo=off
    -mode_ext=2, Intensity_stereo=off, Ms_stereo=on
    -mode_ext=3, Intensity_stereo=on, Ms_stereo=on                */
{
  BYTE temp, i;

  mode_ext=0;

  for (i=0;i<2;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    mode_ext=mode_ext<<1;
    mode_ext=mode_ext | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
```

```
}

void FrameHeader::GoToFinal()
/* Recorrem tots el bits que ens resten del header fins al final */
{
  BYTE i;

  for (i=0;i<4;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    buf_restant[0]=buf_restant[0]-1;
  }
}

void FrameHeader::runErrorCheck()
{
  if (protection==0)
  {
    runBits(16);
  }
}

void FrameHeader::runBits(BYTE nBits)
{
  BYTE i;

  for (i=0;i<nBits;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    buf_restant[0]=buf_restant[0]-1;
  }
}

BYTE FrameHeader::getBitrate()
{
  return (bitrate);
}

BYTE FrameHeader::getSampling()
{
  return (sampling);
}

BYTE FrameHeader::getPadding()
{
  return (padding);
}

BYTE FrameHeader::getID()
{
  return (ID);
```

```
}

BYTE FrameHeader::getMode()
{
  return (mode);
}

BYTE FrameHeader::getModeExt()
{
  return (mode_ext);
}

BYTE FrameHeader::getProtection()
{
  return (protection);
}
```

7.4. Classe SideInfo (*SideInfo.h*)

```
/* Definició de la classe que durà a terme la lectura dels paràmetres
   del side info que ens interessen de cada frame.                  */

class SideInfo
{
  private:
    SideInfoDataStructure SideInfoData;

  public:
    SideInfo(BYTE mode, BYTE id);
    ~SideInfo();
    void setMainDataEnd(BYTE nBits);
    void runBits(BYTE nBits);
    void setScfsi(BYTE channel, BYTE band);
    void setPart23Length(BYTE granule, BYTE channel);
    void setBigValues(BYTE granule, BYTE channel);
    void setGlobalGain(BYTE granule, BYTE channel);
    void setScalefacCompress(BYTE nBits, BYTE granule, BYTE channel);
    void setWindowSwitchingFlag(BYTE granule, BYTE channel);
    void setBlockType(BYTE granule, BYTE channel);
    void setBlockTypeToZero(BYTE granule, BYTE channel);
    void setMixedBlockFlag(BYTE granule, BYTE channel);
    void setTableSelect(BYTE granule, BYTE channel, BYTE region);
    void setSubblockGain(BYTE granule, BYTE channel, BYTE window);
    void setRegion0Count(BYTE granule, BYTE channel);
    void setRegion1Count(BYTE granule, BYTE channel);
    void setPreflag(BYTE granule, BYTE channel);
    void setScalefacScale(BYTE granule, BYTE channel);
    void setCount1tableSelect(BYTE granule, BYTE channel);

    BYTE getWindowSwitchingFlag(BYTE granule, BYTE channel);
    SideInfoDataStructure getSideInfoData();
    DBYTE getGlobalGain(BYTE granule, BYTE channel);
};
```

```
SideInfo::SideInfo(BYTE mode, BYTE id)
{
  BYTE nGranules, nChannels, i, j, z;

  if (mode==3)
    nChannels=1;
  else
    nChannels=2;

  if (id==0)
    nGranules=1;
  else
    nGranules=2;

  /* Anem a recollir tots els paràmetres del Side Info */
  if (id == 1)
  {
    setMainDataEnd(9);
    if (mode == 3)   // En el cas d'un canal
      runBits(5);  // Recorrer bits de Private_bits, en aquest cas 5
    else
      runBits(3);  // Recorrer bits de Private_bits, en aquest cas 3
    for(i=0;i<nChannels;i++)
    {
      for(j=0;j<4;j++)
        setScfsi(j, i);
    }
  }
  else
  {
    setMainDataEnd(8);
    if (mode==3)  // En el cas d'un channel
      runBits(1);  // Recorrer bits de Private_bits, en aquest cas 1
    else         // En el cas de dos channel
      runBits(2);  // Recorrer bits de Private_bits, en aquest cas 2
  }

  for(i=0;i<nGranules;i++)
  {
    for(j=0;j<nChannels;j++)
    {
      setPart23Length(i, j);
      setBigValues(i, j);
      setGlobalGain(i, j);
      if(id==1)
        setScalefacCompress(4, i, j);
      else
        setScalefacCompress(9, i, j);
      setWindowSwitchingFlag(i, j);
      if(getWindowSwitchingFlag(i,j)==1)
      {
        setBlockType(i, j);
        setMixedBlockFlag(i, j);
        for(z=0;z<2;z++)
          setTableSelect(z, i, j);
        for(z=0;z<3;z++)
          setSubblockGain(i, j, z);
      }
      else  // Haig de fer passar els bits que no gasto,
```

```
            // al contrari de quan entro al if.
        {
          setBlockTypeToZero(i, j); // Quan window_switching_flag és 0,
                                    // block_type també.
          for(z=0;z<3;z++)
            setTableSelect(z, i, j);
          setRegion0Count(i, j);
          setRegion1Count(i, j);
        }
        if(id==1)
          setPreflag(i, j);
        setScalefacScale(i, j);
        setCount1tableSelect(i, j);
      }
    }

    if (buf_restant[0] == 0)    // Hem esgotat tots els bits d'aquest byte. Per
                                // deixem preparat el frame per a llegir el byte
                                // que toca (el següent).
    {
      buf_actual[0]++;
      buf_restant[0] = 8;
    }
}

SideInfo::~SideInfo()
{
}

void SideInfo::setMainDataEnd(BYTE nBits)
{
  BYTE temp, i;

  SideInfoData.main_data_end=0;

  for (i=0;i<nBits;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    SideInfoData.main_data_end=SideInfoData.main_data_end<<1;
    SideInfoData.main_data_end=SideInfoData.main_data_end | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}

void SideInfo::runBits(BYTE nBits)
{
  BYTE temp, i;

  for (i=0;i<nBits;i++)
  {
    if (buf_restant[0]==0)
    {
```

```
            buf_actual[0]++;
            buf_restant[0]=8;
          }
          buf_restant[0]=buf_restant[0]-1;
        }
      }

      void SideInfo::setScfsi(BYTE band, BYTE channel)
      {
        BYTE temp;

        if (buf_restant[0]==0)
        {
          buf_actual[0]++;
          buf_restant[0]=8;
        }
        temp=buffer[0][buf_actual[0]];
        temp=temp<<(8-buf_restant[0]);
        temp=temp>>7;
        SideInfoData.scfsi[band][channel]=temp;
        buf_restant[0]=buf_restant[0]-1;

      }

      void SideInfo::setPart23Length(BYTE granule, BYTE channel)
      {
        BYTE temp, i;

        SideInfoData.part2_3_length[granule][channel]=0;

        for (i=0;i<12;i++)
        {
          if (buf_restant[0]==0)
          {
            buf_actual[0]++;
            buf_restant[0]=8;
          }
          temp=buffer[0][buf_actual[0]];
          temp=temp<<(8-buf_restant[0]);
          temp=temp>>7;
          SideInfoData.part2_3_length[granule][channel]=SideInfoData.part2_3_length[granule][channel]<<1;
          SideInfoData.part2_3_length[granule][channel]=SideInfoData.part2_3_length[granule][channel] |
temp;
          buf_restant[0]=buf_restant[0]-1;
        }

      }

      void SideInfo::setBigValues(BYTE granule, BYTE channel)
      {
        BYTE temp, i;

        SideInfoData.big_values[granule][channel]=0;

        for (i=0;i<9;i++)
        {
          if (buf_restant[0]==0)
          {
            buf_actual[0]++;
```

```
        buf_restant[0]=8;
      }
      temp=buffer[0][buf_actual[0]];
      temp=temp<<(8-buf_restant[0]);
      temp=temp>>7;
      SideInfoData.big_values[granule][channel]=SideInfoData.big_values[granule][channel]<<1;
      SideInfoData.big_values[granule][channel]=SideInfoData.big_values[granule][channel] | temp;
      buf_restant[0]=buf_restant[0]-1;
    }
  }


  void SideInfo::setGlobalGain(BYTE granule, BYTE channel)
  {
    BYTE temp, i;

    SideInfoData.global_gain[granule][channel]=0;

    for (i=0;i<8;i++)
    {
      if (buf_restant[0]==0)
      {
        buf_actual[0]++;
        buf_restant[0]=8;
      }
      temp=buffer[0][buf_actual[0]];
      temp=temp<<(8-buf_restant[0]);
      temp=temp>>7;
      SideInfoData.global_gain[granule][channel]=SideInfoData.global_gain[granule][channel]<<1;
      SideInfoData.global_gain[granule][channel]=SideInfoData.global_gain[granule][channel] | temp;
      buf_restant[0]=buf_restant[0]-1;
    }

  }


  void SideInfo::setScalefacCompress(BYTE nBits, BYTE granule, BYTE channel)
  {
    BYTE temp, i;

    SideInfoData.scalefac_compress[granule][channel]=0;

    for (i=0;i<nBits;i++)
    {
      if (buf_restant[0]==0)
      {
        buf_actual[0]++;
        buf_restant[0]=8;
      }
      temp=buffer[0][buf_actual[0]];
      temp=temp<<(8-buf_restant[0]);
      temp=temp>>7;

SideInfoData.scalefac_compress[granule][channel]=SideInfoData.scalefac_compress[granule][channel]<<1;

SideInfoData.scalefac_compress[granule][channel]=SideInfoData.scalefac_compress[granule][channel] | temp;
      buf_restant[0]=buf_restant[0]-1;
    }
  }
```

```
void SideInfo::setWindowSwitchingFlag(BYTE granule, BYTE channel)
{
  BYTE temp;

  if (buf_restant[0]==0)
  {
    buf_actual[0]++;
    buf_restant[0]=8;
  }
  temp=buffer[0][buf_actual[0]];
  temp=temp<<(8-buf_restant[0]);
  temp=temp>>7;
  SideInfoData.window_switching_flag[granule][channel]=temp;
  buf_restant[0]=buf_restant[0]-1;

}


void SideInfo::setBlockType(BYTE granule, BYTE channel)
{
  BYTE temp, i;

  SideInfoData.block_type[granule][channel]=0;

  for (i=0;i<2;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    SideInfoData.block_type[granule][channel]=SideInfoData.block_type[granule][channel]<<1;
    SideInfoData.block_type[granule][channel]=SideInfoData.block_type[granule][channel] | temp;
    buf_restant[0]=buf_restant[0]-1;
  }

}

void SideInfo::setBlockTypeToZero(BYTE granule, BYTE channel)
{
  SideInfoData.block_type[granule][channel] = 0;
}

void SideInfo::setMixedBlockFlag(BYTE granule, BYTE channel)
{
  BYTE temp;

  if (buf_restant[0]==0)
  {
    buf_actual[0]++;
    buf_restant[0]=8;
  }
  temp=buffer[0][buf_actual[0]];
  temp=temp<<(8-buf_restant[0]);
```

```
    temp=temp>>7;
    SideInfoData.mixed_block_flag[granule][channel]=temp;
    buf_restant[0]=buf_restant[0]-1;
  }


  void SideInfo::setTableSelect(BYTE region, BYTE granule, BYTE channel)
  {
    BYTE temp, i;

    SideInfoData.table_select[region][granule][channel]=0;

    for (i=0;i<5;i++)
    {
      if (buf_restant[0]==0)
      {
        buf_actual[0]++;
        buf_restant[0]=8;
      }
      temp=buffer[0][buf_actual[0]];
      temp=temp<<(8-buf_restant[0]);
      temp=temp>>7;

SideInfoData.table_select[region][granule][channel]=SideInfoData.table_select[region][granule][channel]<<1;

SideInfoData.table_select[region][granule][channel]=SideInfoData.table_select[region][granule][channel] | temp;
      buf_restant[0]=buf_restant[0]-1;
    }

  }


  void SideInfo::setSubblockGain(BYTE granule, BYTE channel, BYTE window)
  {
    BYTE temp, i;

    SideInfoData.subblock_gain[granule][channel][window]=0;

    for (i=0;i<3;i++)
    {
      if (buf_restant[0]==0)
      {
        buf_actual[0]++;
        buf_restant[0]=8;
      }
      temp=buffer[0][buf_actual[0]];
      temp=temp<<(8-buf_restant[0]);
      temp=temp>>7;

SideInfoData.subblock_gain[granule][channel][window]=SideInfoData.subblock_gain[granule][channel][windo
w]<<1;

SideInfoData.subblock_gain[granule][channel][window]=SideInfoData.subblock_gain[granule][channel][windo
w] | temp;
      buf_restant[0]=buf_restant[0]-1;
    }
  }
```

```
void SideInfo::setRegion0Count(BYTE granule, BYTE channel)
{
  BYTE temp, i;

  SideInfoData.region0_count[granule][channel]=0;

  for (i=0;i<4;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    SideInfoData.region0_count[granule][channel]=SideInfoData.region0_count[granule][channel]<<1;
    SideInfoData.region0_count[granule][channel]=SideInfoData.region0_count[granule][channel] | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}


void SideInfo::setRegion1Count(BYTE granule, BYTE channel)
{
  BYTE temp, i;

  SideInfoData.region1_count[granule][channel]=0;

  for (i=0;i<3;i++)
  {
    if (buf_restant[0]==0)
    {
      buf_actual[0]++;
      buf_restant[0]=8;
    }
    temp=buffer[0][buf_actual[0]];
    temp=temp<<(8-buf_restant[0]);
    temp=temp>>7;
    SideInfoData.region1_count[granule][channel]=SideInfoData.region1_count[granule][channel]<<1;
    SideInfoData.region1_count[granule][channel]=SideInfoData.region1_count[granule][channel] | temp;
    buf_restant[0]=buf_restant[0]-1;
  }
}


void SideInfo::setPreflag(BYTE granule, BYTE channel)
{
  BYTE temp;

  if (buf_restant[0]==0)
  {
    buf_actual[0]++;
    buf_restant[0]=8;
  }
  temp=buffer[0][buf_actual[0]];
  temp=temp<<(8-buf_restant[0]);
  temp=temp>>7;
  SideInfoData.preflag[granule][channel]=temp;
```

```
  buf_restant[0]=buf_restant[0]-1;
}


void SideInfo::setScalefacScale(BYTE granule, BYTE channel)
{
  BYTE temp;

  if (buf_restant[0]==0)
  {
    buf_actual[0]++;
    buf_restant[0]=8;
  }
  temp=buffer[0][buf_actual[0]];
  temp=temp<<(8-buf_restant[0]);
  temp=temp>>7;
  SideInfoData.scalefac_scale[granule][channel]=temp;
  buf_restant[0]=buf_restant[0]-1;
}


void SideInfo::setCount1tableSelect(BYTE granule, BYTE channel)
{
  BYTE temp;

  if (buf_restant[0]==0)
  {
    buf_actual[0]++;
    buf_restant[0]=8;
  }
  temp=buffer[0][buf_actual[0]];
  temp=temp<<(8-buf_restant[0]);
  temp=temp>>7;
  SideInfoData.count1table_select[granule][channel]=temp;
  buf_restant[0]=buf_restant[0]-1;
}

BYTE SideInfo::getWindowSwitchingFlag(BYTE granule, BYTE channel)
{
  return (SideInfoData.window_switching_flag[granule][channel]);
}

DBYTE SideInfo::getGlobalGain(BYTE granule, BYTE channel)
{
  return (SideInfoData.global_gain[granule][channel]);
}

SideInfoDataStructure SideInfo::getSideInfoData()
{
  return (SideInfoData);
}
```

7.5. Classe MainData (*MainData.h*)

```
/* Definició de la classe que durà a terme la lectura de les
   dades del main data.                                    */

#include <math.h>
#include "huffman.h"


class MainData
{
  private:
    BYTE scalefac[21][3][2][2];
    BYTE scalefac_l[2][21];
    BYTE scalefac_s[2][12][3];
    BYTE HuffmanCodeBits[512];
    float Dequantized[2][2][576];
    DBYTE hcb_pointer;
    BYTE hcb_bit;
    SideInfoDataStructure sid;

    BYTE data[512];
    DBYTE data_pointer;
    DBYTE data_actual;
    BYTE data_restant;

  public:
    MainData(SideInfoDataStructure sid, BYTE mode, BYTE mode_ext, BYTE id);
    ~MainData();
    void setScalefacAndHuffmanMPEG1(BYTE nCh);
    void setScalefacAndHuffmanMPEG2(BYTE nCh, BYTE mode_extension);
    void setScalefac(BYTE cb, BYTE gr, BYTE ch);
    void setScalefac(BYTE cb, BYTE window, BYTE gr, BYTE ch);
    void setHuffmanCodeBits(BYTE gr, BYTE ch, BYTE id, BYTE mode_extension);
    DBYTE getPart2LengthMPEG1(BYTE gr, BYTE ch);
    DBYTE getPart2LengthMPEG2(BYTE gr, BYTE ch, BYTE mode_extension);
    BYTE getScalefacLen(BYTE cb, BYTE gr, BYTE ch);
    void getMainData(DBYTE mde);
    void runHuffmanAndDequantizer(BYTE granule, BYTE channel);
    void setDequantizedValues(BYTE granule, BYTE channel, DBYTE is[576]);
    float Dequantizer_exp_long(int sfb, BYTE granule, BYTE channel);
    float Dequantizer_exp_short(int sfb, BYTE granule, BYTE window)
};

MainData::MainData(float ***values, SideInfoDataStructure s, BYTE mode, BYTE mode_ext, BYTE id)
{
  BYTE nChannels;

  values=Dequantized;
  if (mode==3)
    nChannels=1;
  else
    nChannels=2;
```

```
  sid = s;

  /* Agafem tot els bits del main_data amb l'ajuda del punter main_data_end */
  data_pointer=0;
  data_actual=0;
  data_restant=0;

  getMainData(sid.main_data_end);

  /* Ara anem a agafar els scalefactors i els bits codificats en Huffman */
  if (id == 1)     // Es MPEG 1
    setScalefacAndHuffmanMPEG1(nChannels);
  else             // Es MPEG 2
    setScalefacAndHuffmanMPEG2(nChannels, mode_ext);

}

MainData::~MainData()
{
}

void MainData::setScalefacAndHuffmanMPEG1(BYTE nCh)
{
  DBYTE i;
  BYTE switch_point_l[2][2], switch_point_s[2][2];
  BYTE cb, window, gr, ch;

  for (gr=0;gr<2;gr++)
  {
    for (ch=0 ; ch<nCh ; ch++)
    {
      if (sid.mixed_block_flag[gr][ch]==0)
      {
        switch_point_l[gr][ch]=0;
        switch_point_s[gr][ch]=0;
      }
      else
      {
        switch_point_l[gr][ch]=8;
        switch_point_s[gr][ch]=3;
      }
    }
  }

  for (gr=0 ; gr<2 ; gr++)
  {
    for (ch=0 ; ch<nCh ; ch++)
    {
      if (sid.window_switching_flag[gr][ch]==1 && sid.block_type[gr][ch]==2)
      {
        for (cb=0 ; cb<switch_point_l[gr][ch] ; cb++)
        {
          if (sid.scfsi[ch][cb]==0 || gr==0)
            setScalefac(cb, gr, ch);
          else if (gr==1 && sid.scfsi[ch][cb]==1)
            scalefac[cb][0][gr][ch] = scalefac[cb][0][0][ch];
        }

        for (cb=switch_point_s[gr][ch] ; cb<12 ; cb++)
```

```
        {
          for (window=0 ; window<3 ; window++)
          {
            if (sid.scfsi[ch][cb]==0 || gr==0)
              setScalefac(cb, window, gr, ch);
            else if (gr==1 && sid.scfsi[ch][cb]==1)
              scalefac[cb][window][gr][ch] = scalefac[cb][window][0][ch];
          }
        }
      }
      else
      {
        for (cb=0 ; cb<21 ; cb++)
        {
          if (sid.scfsi[ch][cb]==0 || gr==0)
            setScalefac(cb, gr, ch);
          else if (gr==1 && sid.scfsi[ch][cb]==1)
            scalefac[cb][0][gr][ch] = scalefac[cb][0][0][ch];
        }
      }

      /* Agafem els bits codificats en Huffman */
      setHuffmanCodeBits(gr, ch, 1, 0);
    }

  }

 /* Aquí es situarien els ancilliary bits, concretament d'aquí fins al
    final del main_data, marcat pel main_data_end.
    En el nostre cas no necessitem agafar aquests bits ja que no tenen
    cap utilitat per a la nostra aplicació.                    */
}

void MainData::setScalefacAndHuffmanMPEG2(BYTE nCh, BYTE mode_extension)
{
  BYTE ch, sfb, window;

  for (ch=0; ch<nCh; ch++)
  {
    if ((sid.window_switching_flag[0][ch]==1) && (sid.block_type[0][ch]==2))
    {
      if (sid.mixed_block_flag[0][ch])
      {
        for (sfb=0; sfb<8; sfb++)
          scalefac_l[ch][sfb];
        for (sfb=3; sfb<12; sfb++)
        {
          for (window=0; window<3; window++)
            scalefac_s[ch][sfb][window];
                    }
      }
      else
      {
        for (sfb=0; sfb<12; sfb++)
        {
          for (window=0; window<3; window++)
            scalefac_s[ch][sfb][window];
                    }
      }
```

```
      }
      else
      {
        for (sfb=0;sfb<21;sfb++)
          scalefac_l[ch][sfb];
      }

      /* Agafem els bits codificats en Huffman */
      setHuffmanCodeBits(0, ch, 0, mode_extension);

    }

   /* Aquí es situarien els ancilliary bits, concretament d'aquí fins al
      final del main_data, marcat pel main_data_end.
      En el nostre cas no necessitem agafar aquests bits ja que no tenen
      cap utilitat per a la nostra aplicació.                    */

}

BYTE MainData::getScalefacLen(BYTE cb, BYTE gr, BYTE ch)
{
  BYTE scalefac_len[2][16] = {{0,0,0,0,3,1,1,1,2,2,2,3,3,3,4,4},
                              {0,1,2,3,0,1,2,3,1,2,3,1,2,3,2,3}};
  BYTE slen, slen1, slen2;

  slen1=scalefac_len[0][sid.scalefac_compress[gr][ch]];
  slen2=scalefac_len[1][sid.scalefac_compress[gr][ch]];

  if (sid.block_type[gr][ch]==2)
  {
    if (sid.mixed_block_flag[gr][ch]==0)
    {
      if (cb<6)    // Es tracta d'una scalefactor en les bandes 0-5
        slen=slen1;
      else         // Es tracta d'una scalefactor en les bandes 6-11
        slen=slen2;
    }
    else
    {
      if (cb<8)    // Es tracta d'una scalefactor en les bandes 0-7 (long
                   // window scalefactor band) i 3-5 (short window scalefactor band)

        slen=slen1;
      else         // Es tracta d'una scalefactor en les bandes 6-11
        slen=slen2;
    }
  }
  else
  {
    if (cb<11)     // Es tracta d'una scalefactor en les bandes 0-10
      slen=slen1;
    else           // Es tracta d'una scalefactor en les bandes 11-20
      slen=slen2;
  }

  return (slen);
}

void MainData::setScalefac(BYTE cb, BYTE gr, BYTE ch)
```

```
{
  BYTE scalefac_len;
  BYTE temp, i;

  scalefac_len = getScalefacLen(cb, gr, ch);

  scalefac[cb][0][gr][ch]=0;

  for (i=0;i<scalefac_len;i++)
  {
    if (data_restant==0)
    {
      data_actual++;
      data_restant=8;
    }
    temp=data[data_actual];
    temp=temp<<(8-data_restant);
    temp=temp>>7;
    scalefac[cb][0][gr][ch]=scalefac[cb][0][gr][ch]<<1;
    scalefac[cb][0][gr][ch]=scalefac[cb][0][gr][ch] | temp;
    data_restant=data_restant-1;
  }
}

void MainData::setScalefac(BYTE cb, BYTE window, BYTE gr, BYTE ch)
{
  BYTE scalefac_len;
  BYTE temp, i;

  scalefac_len = getScalefacLen(cb, gr, ch);

  scalefac[cb][window][gr][ch]=0;

  for (i=0;i<scalefac_len;i++)
  {
    if (data_restant==0)
    {
      data_actual++;
      data_restant=8;
    }
    temp=data[data_actual];
    temp=temp<<(8-data_restant);
    temp=temp>>7;
    scalefac[cb][window][gr][ch]=scalefac[cb][window][gr][ch]<<1;
    scalefac[cb][window][gr][ch]=scalefac[cb][window][gr][ch] | temp;
    data_restant=data_restant-1;
  }
}

void MainData::getMainData(DBYTE mde)
{
  BYTE main_data_beg;    // Indica el buffer (frame) en el que comença
                         // el main_data del frame que estem tractant.
  DBYTE i, j;
  DBYTE main_data_end, main_data_end_pos;
  BOOL trobat;

  /* Busquem primer on estaria el main_data_begin.
     Estaria en el final més 1 bit del main_data del frame anterior.
```

```
   Anem a buscar-lo mirant els buffer anteriors on el punter buf_actual
   no havia arribat al valor del buf_pointer (final del frame).        */
for (i=0 ; i<9 ; i++)
{
  if (buf_actual[i] < buf_pointer[i]) // Aquest frame conté el principi
                                      // del main_data del nostre actual.
    main_data_beg = i;
}
/* Ara anem a buscar on estarà el final del main_data. Main_data_end
   contindrà el buffer (frame) on es troba el final del main_data i en
   el punter buf_actual del buffer hi trobarem el punt exacte dins d'ell. */
main_data_end = mde;
trobat=0;
for (i=0 ; i<9 && !trobat ; i++)
{
  if (main_data_end == 0)  // El main_data_end està al final del buffer
                           // (frame) que em tractat abans o és el primer.
  {
    if (i == 0)  // Està en el primer que tractem, o sigui l'actual. Per
                 // tant el main_data anirà des del final del side info
                 // actual fins al final del frame actual.
    {
      main_data_end = i;
      main_data_end_pos = buf_pointer[i];
      trobat=1;
    }
  }
  else
  {
    if ((buf_pointer[i]-buf_actual[i]) <= main_data_end)
    {
      main_data_end = main_data_end - (buf_pointer[i]-buf_actual[i]);
    }
    else  // Estarà en aquest frame
    {
      main_data_end_pos = buf_pointer[i] - main_data_end;
            // Aquest punter apunta a la posició dins del buffer
            // (frame) on hi haurà el final del main_data.
      main_data_end = i;   // Posem a main_data_end el buffer que conté el
                           // frame amb el final del main_data de l'actual.
      trobat=1;
    }
  }
}

/* Ara que sabem el principi i el final del main_data, ja podem agafar-lo. */
for (i=main_data_beg ; i>main_data_end ; i--)
{
  for (j=buf_actual[i] ; j<buf_pointer[i] ; j++)
  {
    data[data_pointer] = buffer[i][j];
    data_pointer = data_pointer+1;
  }

  buf_actual[i]= buf_pointer[i];
}

for (j=buf_actual[i] ; j<main_data_end_pos ; j++)
{
```

```
    data[data_pointer] = buffer[i][j];
    data_pointer = data_pointer+1;
  }

  if (main_data_end_pos == buf_pointer[i]) // El main_data ocupa fins el
                                  // final exacte del frame
    buf_actual[i] = buf_pointer[i];
  else
    buf_actual[i]= main_data_end_pos-1;
}

DBYTE MainData::getPart2LengthMPEG1(BYTE gr, BYTE ch)
{
  BYTE scalefac_len[2][16] = {{0,0,0,0,3,1,1,1,2,2,2,3,3,3,4,4},
                      {0,1,2,3,0,1,2,3,1,2,3,1,2,3,2,3}};
  BYTE slen1, slen2;
  DBYTE part2;

  /* Calculem el nombre de bits que han ocupat els scalefactors */
  slen1=scalefac_len[0][sid.scalefac_compress[gr][ch]];
  slen2=scalefac_len[1][sid.scalefac_compress[gr][ch]];

  if (sid.mixed_block_flag[gr][ch] == 0)
  {
    if (sid.block_type[gr][ch] == 2)   // Short blocks
      part2 = 18 * slen1 + 18 * slen2;
    else   // Long blocks
      part2 = 11 * slen1 + 10 * slen2;
  }
  else
  {
    if (sid.block_type[gr][ch] == 2)   // Short blocks
      part2 = 17 * slen1 + 18 * slen2;
    else   // Long blocks
      part2 = 11 * slen1 + 10 * slen2;
  }

  return (part2);
}

DBYTE MainData::getPart2LengthMPEG2(BYTE gr, BYTE ch, BYTE mode_extension)
{
  DBYTE intensity_scale[2][2], int_scalefac_compress[2][2];
  DBYTE part2, slen1, slen2, slen3, slen4;
  BYTE nr_of_sfb1, nr_of_sfb2, nr_of_sfb3, nr_of_sfb4;

  /* Calculem el nombre de bits que han ocupat els scalefactors */
  if (!(((mode_extension==1)||(mode_extension==3)) && (ch==1)))
  {
    if(sid.scalefac_compress[0][ch]<400)
    {
      slen1=(sid.scalefac_compress[gr][ch]>>4)/5;
            slen2=(sid.scalefac_compress[gr][ch]>>4)%5;
            slen3=(sid.scalefac_compress[gr][ch]%16)>>2;
            slen4=sid.scalefac_compress[gr][ch]%4;
            sid.preflag[gr][ch]=0;
            if(sid.block_type[gr][ch]!=2)
      {
                    nr_of_sfb1=6;
```

```
                nr_of_sfb2=5;
                nr_of_sfb3=5;
                nr_of_sfb4=5;
        }
        else
 {
                if(sid.mixed_block_flag[gr][ch]==0)
   {
                        nr_of_sfb1=9;
                        nr_of_sfb2=9;
                        nr_of_sfb3=9;
                        nr_of_sfb4=9;
                }
                else
   {
                        nr_of_sfb1=6;
                        nr_of_sfb2=9;
                        nr_of_sfb3=9;
                        nr_of_sfb4=9;
                 }
        }
}
if(sid.scalefac_compress[gr][ch]>=400 && sid.scalefac_compress[gr][ch]<500)
 {
        slen1=((sid.scalefac_compress[gr][ch]-400)>>2)/5;
        slen2=((sid.scalefac_compress[gr][ch]-400)>>2)%5;
        slen3=(sid.scalefac_compress[gr][ch]-400)%4;
        slen4=0;
        sid.preflag[gr][ch]=0;
        if(sid.block_type[gr][ch]!=2)
  {
                nr_of_sfb1=6;
                nr_of_sfb2=5;
                nr_of_sfb3=7;
                nr_of_sfb4=3;
        }
        else
  {
                if(sid.mixed_block_flag[gr][ch]==0)
   {
                        nr_of_sfb1=9;
                        nr_of_sfb2=9;
                        nr_of_sfb3=12;
                        nr_of_sfb4=6;
                }
                else
   {
                        nr_of_sfb1=6;
                        nr_of_sfb2=9;
                        nr_of_sfb3=12;
                        nr_of_sfb4=6;
                 }
        }
}
if(sid.scalefac_compress[gr][ch]>=500 && sid.scalefac_compress[gr][ch]<512)
 {
        slen1=(sid.scalefac_compress[gr][ch]-500)/3;
        slen2=(sid.scalefac_compress[gr][ch]-500)%3;
        slen3=0;
```

```
            slen4=0;
            sid.preflag[gr][ch]=1;
            if(sid.block_type[gr][ch]!=2)
      {
                        nr_of_sfb1=11;
                        nr_of_sfb2=10;
                        nr_of_sfb3=0;
                        nr_of_sfb4=0;
            }
            else
      {
                        if(sid.mixed_block_flag[gr][ch]==0)
        {
                                    nr_of_sfb1=18;
                                    nr_of_sfb2=18;
                                    nr_of_sfb3=0;
                                    nr_of_sfb4=0;
                        }
                        else
        {
                                    nr_of_sfb1=15;
                                    nr_of_sfb2=18;
                                    nr_of_sfb3=0;
                                    nr_of_sfb4=0;
                         }
            }
    }
}
else
{
    intensity_scale[gr][ch]=sid.scalefac_compress[gr][ch]%2;
    int_scalefac_compress[gr][ch]=sid.scalefac_compress[gr][ch]>>1;
    if(int_scalefac_compress[gr][ch]<180)
     {
            slen1=int_scalefac_compress[gr][ch]/36;
            slen2=(int_scalefac_compress[gr][ch]%36)/6;
            slen3=(int_scalefac_compress[gr][ch]%36)%6;
            slen4=0;
            sid.preflag[gr][ch]=0;
            if(sid.block_type[gr][ch]!=2)
      {
                        nr_of_sfb1=7;
                        nr_of_sfb2=7;
                        nr_of_sfb3=7;
                        nr_of_sfb4=0;
            }
            else
      {
                        if(sid.mixed_block_flag[gr][ch]==0)
        {
                                    nr_of_sfb1=12;
                                    nr_of_sfb2=12;
                                    nr_of_sfb3=12;
                                    nr_of_sfb4=0;
                        }
                        else
        {
                                    nr_of_sfb1=6;
                                    nr_of_sfb2=15;
```

```
                        nr_of_sfb3=12;
                        nr_of_sfb4=0;
                    }
        }
}
if(int_scalefac_compress[gr][ch]>=180 && int_scalefac_compress[gr][ch]<244)
 {
        slen1=((int_scalefac_compress[gr][ch]-180)%64)>>4;
        slen2=((int_scalefac_compress[gr][ch]-180)%16)>>2;
        slen3=(int_scalefac_compress[gr][ch]-180)%4;
        slen4=0;
        sid.preflag[gr][ch]=0;
        if(sid.block_type[gr][ch]!=2)
  {
                nr_of_sfb1=6;
                nr_of_sfb2=6;
                nr_of_sfb3=6;
                nr_of_sfb4=3;
        }
        else
  {
                if(sid.mixed_block_flag[gr][ch]==0)
    {
                        nr_of_sfb1=12;
                        nr_of_sfb2=9;
                        nr_of_sfb3=9;
                        nr_of_sfb4=6;
                }
                else
    {
                        nr_of_sfb1=6;
                        nr_of_sfb2=12;
                        nr_of_sfb3=9;
                        nr_of_sfb4=6;
          }
        }
}
if(int_scalefac_compress[gr][ch]>=244 && int_scalefac_compress[gr][ch]<=255)
 {
        slen1=(int_scalefac_compress[gr][ch]-244)/3;
        slen2=(int_scalefac_compress[gr][ch]-244)%3;
        slen3=0;
        slen4=0;
        sid.preflag[gr][ch]=0;
        if(sid.block_type[gr][ch]!=2)
  {
                nr_of_sfb1=8;
                nr_of_sfb2=8;
                nr_of_sfb3=5;
                nr_of_sfb4=0;
        }
        else
  {
                if(sid.mixed_block_flag[gr][ch]==0)
    {
                        nr_of_sfb1=15;
                        nr_of_sfb2=12;
                        nr_of_sfb3=9;
                        nr_of_sfb4=0;
```

```
                    }
                    else
        {
                        nr_of_sfb1=6;
                        nr_of_sfb2=18;
                        nr_of_sfb3=9;
                        nr_of_sfb4=0;
                    }
            }
    }
  }

  part2=nr_of_sfb1*slen1+nr_of_sfb2*slen2+nr_of_sfb3*slen3+nr_of_sfb4*slen4;

  return (part2);
}

void MainData::setHuffmanCodeBits(BYTE gr, BYTE ch, BYTE id, BYTE mode_extension)
/* Retorna el nombre de bytes codificats en Huffman */
{
  DBYTE length, i, part2_length;
  BYTE temp;

  hcb_bit=8;
  hcb_pointer=0;

  if (id == 1)   // Es MPEG 1
    part2_length = getPart2LengthMPEG1(gr, ch);
  else          // Es MPEG 2
    part2_length = getPart2LengthMPEG2(gr, ch, mode_extension);

  /* Anem a calcular el nombre de bits que ocuparàn els bits codificats
    en Huffman.                                          */
  length = sid.part2_3_length[gr][ch] - part2_length;

  for (i=0 ; i<length ; i++)
  {
    if (hcb_bit==0)
    {
      hcb_pointer++;
      hcb_bit=8;
    }

    if (data_restant==0)
    {
      data_actual++;
      data_restant=8;
    }

    temp = data[data_actual];
    temp = temp << (8-data_restant);
    temp = temp >> 7;
    HuffmanCodeBits[hcb_pointer] = HuffmanCodeBits[hcb_pointer] << 1;
    HuffmanCodeBits[hcb_pointer] = HuffmanCodeBits[hcb_pointer] | temp;
    hcb_bit = hcb_bit - 1;
    data_restant = data_restant - 1;
  }

  /* Ja podem decodificar els bits codificats amb l'algorisme de Huffman */
```

```
  runHuffmanAndDequantizer(gr, ch);
}

void MainData::runHuffmanAndDequantizer(BYTE granule, BYTE channel)
{
   /* Decodificarem els bits codificats amb l'algorisme de Huffman.
      Seguirem el següent procediment: decodificar els bits
      necessaris per a esgotar els parells de línies marcats per
      big_values amb l'ajuda de les taules de table_select, quan
      s'acaven, esgotar els bits restants fins arribar a les 576
      línies o al final dels Huffman code bits amb l'ajuda de les
      taules de count1table_select. */

   DBYTE i, linia, is[576], pairs;
   BYTE region, hcb_restant=8, codedlen, taula, trobat, tempbyte;
   DBYTE hufByte=0, codedvalue, valuex, valuey, value, linbits, temp;
   BYTE max_valuexy_for_table[32] = {1,2,3,3,0,4,4,6,6,6,8,8,8,16,0,16,16,16,
                                     16,16,16,16,16,16,16,16,16,16,16,16,16,16};

   linia=0;
   for (pairs=0 ; pairs<sid.big_values[granule][channel] ; pairs++)
   {
     /* Comprobem a quina regió pertany aquesta línia */
     if (pairs<sid.region0_count[granule][channel])
       region=0;
     else if (pairs>sid.region0_count[granule][channel] &&
           (pairs-sid.region0_count[granule][channel])<sid.region1_count[granule][channel])
       region=1;
     else
       region=2;

     /* Agafem la taula amb la que decodifiquem els següents bits */
     taula = sid.table_select[region][granule][channel];

     /* Comencem a agafar bits i comparar-los amb els hcod i hlen de les
       taules per a veure si hem trobat els valors per x i y           */
     codedlen=0;
     codedvalue=0;
     trobat=0;
     while (!trobat)
     {
       if (hcb_restant==0)
       {
         hufByte++;
         hcb_restant=8;
       }

       /* Agafem el possible codi/llargada codificat amb la taula */
       codedlen++;
       tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
       temp = tempbyte >> 7;

       codedvalue = codedvalue << 1;
       codedvalue = codedvalue | temp;

       /* Comparem el valor de codedvalue amb els hcod de la taula
         indicada per la variable table_select[region][granule][channel] */
       for (valuex=0 ; valuex<max_valuexy_for_table[taula] && !trobat; valuex++)
       {
```

```
    for (valuey=0 ; valuey<max_valuexy_for_table[taula] && !trobat; valuey++)
    {
      if (hcod[taula][valuex][valuey] == codedvalue &&
          hlen[taula][valuex][valuey] == codedlen)
        /* Hem trobat el valor */
        trobat=1;
    }
  }

  hcb_restant--;
}

/* Llegim el linbitsx i l'afegim a la x. */
linbits=0;
for (i=0 ; i<16 ; i++)
{
  if (hcb_restant==0)
  {
    hufByte++;
    hcb_restant=8;
  }

  tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
  temp = tempbyte >> 7;
  linbits = linbits << 1;
  linbits =  linbits | temp;

  hcb_restant--;
}
valuex = valuex - 1;
valuex = valuex + linbits;

/* Llegim el signx en el cas de que s'hagi transmès */
if (valuex != 0)  // S'ha transmès
{
  if (hcb_restant==0)
  {
    hufByte++;
    hcb_restant=8;
  }
  tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
  temp = tempbyte >> 7;

  /* Multipliquem per -1 si el bit de signe és 1, sino queda igual */
  if (temp == 1)
    valuex = valuex * (-1);

  hcb_restant--;
}

/* Llegim el linbitsy i l'afegim a la y. */
linbits=0;
for (i=0 ; i<16 ; i++)
{
  if (hcb_restant==0)
  {
    hufByte++;
    hcb_restant=8;
  }
```

```
      tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
      temp = tempbyte >> 7;
      linbits = linbits << 1;
      linbits =  linbits | temp;

      hcb_restant--;
    }
    valuey = valuey - 1;
    valuey = valuey + linbits;

    /* Llegim el signy en el cas de que s'hagi transmès */
    if (valuey != 0)  // S'ha transmès
    {
      if (hcb_restant==0)
      {
        hufByte++;
        hcb_restant=8;
      }
      tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
      temp = tempbyte >> 7;

      /* Multipliquem per -1 si el bit de signe és 1, sino queda igual */
      if (temp == 1)
        valuey = valuey * (-1);

      hcb_restant--;
    }

    /* Ja podem emmagatzemar les línies de freqüència quantificades */
    is[linia] = valuex;
    linia++;
    is[linia] = valuey;
    linia++;
  }

  /* Decodifiquem les línies que queden fins arribar al final amb la
    taula count1table_select[granule][channel].                    */
  while (!(linia>=576 && hufByte<hcb_pointer && hcb_restant>hcb_bit))
  {
    /* Agafem la taula amb la que decodifiquem els següents bits */
    taula = sid.count1table_select[granule][channel];

    /* Comencem a agafar bits i comparar-los amb els hqcod i hqlen de les
      taules per a veure si hem trobat els valors per v,w,x i y         */
    codedlen=0;
    codedvalue=0;
    trobat=0;
    while (!trobat)
    {
      if (hcb_restant==0)
      {
        hufByte++;
        hcb_restant=8;
      }

      /* Agafem el possible codi/llargada codificat amb la taula */
      codedlen++;
      tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
```

```
  temp = tempbyte >> 7;

  codedvalue = codedvalue << 1;
  codedvalue = codedvalue | temp;

  /* Comparem el valor de codedvalue amb els hqcod de la taula indicada
     per la variable count1table_select[granule][channel] */
  for (value=0 ; value<16 && !trobat; value++)
  {
    if (hqcod[taula][value] == codedvalue &&
        hqlen[taula][value] == codedlen)
       /* Hem trobat el valor */
       trobat=1;
  }

  hcb_restant--;
}

/* Ja podem emmagatzemar les línies */

/* Primer la del valor v */
temp = value >> 3;
/* Llegim el signv en el cas de que s'hagi transmès */
if (temp != 0)  // S'ha transmès
{
  if (hcb_restant==0)
  {
    hufByte++;
    hcb_restant=8;
  }
  tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
  temp = tempbyte >> 7;
  /* Multipliquem per -1 si el bit de signe és 1, sino queda igual */
  if (temp == 1)
    temp = temp * (-1);
  hcb_restant--;
}
/* Ja podem emmagatzemar la línia amb el signe adequat */
is[linia] = temp;
linia++;

/* Després la del valor w */
tempbyte = value << 1;
temp = tempbyte >> 3;
/* Llegim el signw en el cas de que s'hagi transmès */
if (temp != 0)  // S'ha transmès
{
  if (hcb_restant==0)
  {
    hufByte++;
    hcb_restant=8;
  }
  tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
  temp = tempbyte >> 7;
  /* Multipliquem per -1 si el bit de signe és 1, sino queda igual */
  if (temp == 1)
    temp = temp * (-1);
  hcb_restant--;
}
```

```
      /* Ja podem emmagatzemar la línia amb el signe adequat */
      is[linia] = temp;
      linia++;

      /* Després la del valor x */
      tempbyte = value << 2;
      temp = tempbyte >> 3;
      /* Llegim el signx en el cas de que s'hagi transmès */
      if (temp != 0)  // S'ha transmès
      {
        if (hcb_restant==0)
        {
          hufByte++;
          hcb_restant=8;
        }
        tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
        temp = tempbyte >> 7;
        /* Multipliquem per -1 si el bit de signe és 1, sino queda igual */
        if (temp == 1)
          temp = temp * (-1);
        hcb_restant--;
      }
      /* Ja podem emmagatzemar la línia amb el signe adequat */
      is[linia] = temp;
      linia++;

      /* Per últim la del valor y */
      tempbyte = value << 3;
      temp = tempbyte >> 3;
      /* Llegim el signy en el cas de que s'hagi transmès */
      if (temp != 0)  // S'ha transmès
      {
        if (hcb_restant==0)
        {
          hufByte++;
          hcb_restant=8;
        }
        tempbyte = HuffmanCodeBits[hufByte] << (8-hcb_restant);
        temp = tempbyte >> 7;
        /* Multipliquem per -1 si el bit de signe és 1, sino queda igual */
        if (temp == 1)
          temp = temp * (-1);
        hcb_restant--;
      }
      /* Ja podem emmagatzemar la línia amb el signe adequat */
      is[linia] = temp;
      linia++;
    }

    /* Ja podem posar en marxa el procés de desquantificació */
    setDequantizedValues(granule, channel, is);
}

float Dequantizer_exp_long(int sfb, BYTE granule, BYTE channel)
{
    /* Càlcul de la segona part de la fórmula de desquantificació
       per long blocks.                                          */
    BYTE pretab[] = {0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,2};  // Preemphasis
    float a;
```

```
  a = 2^((1/4) * (sid.global_gain[granule][channel] - 210)) *
     2^-(((sid.scalefac_scale[granule][channel] + 1) / 2) *
     (scalefac_l[channel][sfb] + sid.preflag[granule][channel] * pretab[sfb]);

  return (a);
}

float Dequantizer_exp_short(int sfb, BYTE granule, BYTE channel, BYTE window)
{
  /* Càlcul de la segona part de la fórmula de desquantificació
     per short blocks.                                          */
  BYTE pretab[] = {0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,2};  // Preemphasis
  float a;

  a = 2^((1/4) * (sid.global_gain[granule][channel] - 210 - 8 *
     sid.subblock_gain[granule][channel][window])) *
     2^-(((sid.scalefac_scale[granule][channel] + 1) / 2) *
     scalefac_s[channel][sfb][window]);

  return (a);
}

void MainData::setDequantizedValues(BYTE granule, BYTE channel, DBYTE is[576])
{
  BYTE window,window_len;
  int l,sfb;
  float a;

  /* Calculem amb la fórmula de la desquantificació */

  if (sid.window_switching_flag[granule][channel] && sid.block_type[granule][channel]==2)
  {
    if (sid.mixed_block_flag[granule][channel])
    {
      /* Comencem amb la part dels mixed blocks/long blocks  */
      l=0;
      sfb=0;
      a=Dequantizer_exp_long(sfb, granule, channel);
      while (l<36)
      {
        /* Emmagatzemem el component desquantificat */
        Dequantized[ch][0][l]=sign(is[l]) * abs(is[l])^(4/3) * a;
        if (l==t_l[sfb])
        {
          sfb++;
          a=Dequantizer_exp_long(sfb, granule, channel);
        }
        l++;
      }

      /* Comença la part dels mixed blocks/short blocks */
      sfb=3;
      window_len=t_s[sfb]-t_s[sfb-1];
      while (l<(sid.big_values[granule][channel]*2+count1[granule][channel]*4))
      {
        for (window=0;window<3;window++)
        {
          a=Dequantizer_exp_short(sfb, granule, channel, window);
```

```
            for (i=0;i<window_len;i++)
            {
              Dequantized[ch][0][l]=sign(is[l]) * abs(is[l])^(4/3) * a;
              l++;
            }
          }
          sfb++;
          window_len=t_s[sfb]-t_s[sfb-1];
        }
        while (l<576)
          Dequantized[ch][0][l++]=0;
      }
      else
      {
        /* Comença la part dels short blocks  */
        sfb=0; l=0;
        window_len=t_s[0]+1;
        while (l<(sid.big_values[granule][channel]*2+count1[granule][channel]*4))
        {
          for (window=0;window<3;window++)
          {
            a=Dequantizer_exp_short(sfb, granule, channel, window);
            for (i=0;i<window_len;i++)
            {
              Dequantized[ch][0][l]=sign(is[l]) * abs(is[l])^(4/3) * a;
              l++;
            }
        }
        sfb++;
        window_len=t_s[sfb]-t_s[sfb-1];
      }
      while (l<576)
        Dequantized[ch][0][l++]=0;
    }
    else
    {
      /* Comença la part dels long blocks */
      sfb=0; l=0;
      a=Dequantizer_exp_long(sfb, granule, channel);
      while (l<non_zero[ch])
      {
        Dequantized[ch][0][l]=sign(is[l]) * abs(is[l])^(4/3) * a;
        if (l==t_l[sfb])
        {
          sfb++;
          a=Dequantizer_exp_long(sfb, granule, channel);
        }
        l++;
      }
      while (l<576)
        Dequantized[ch][0][l++]=0;
    }
  }
```

7.6. Altres arxius

7.6.1. *Huffman.h*

```
/* S'inclouen les taules de Huffman necessaries per a dur a terme
   el procés de decodificació dels Huffman code bits.
   La taula htables conté els valors cadascuna de les 32 taules.
   La taula hlen conté el tamany del valor representat en la taula corresponent.
   En les dues taules, el primer índex és el nombre de taula, el segon és
   el valor de x i el tercer és el valor de la y.
   La taula hqtables conté els valors de les dues taules per quadruples.
   La taula hqlen conté el tamany del valor representat en la taula de quadruples corresponent.
   En les dues taules, el primer índex és el nombre de taula, el segon és
   el valor del quadruple (v-w-x-y).                                        */


DBYTE hcod[32][16][16] = {
   /* Taula 0 */
   {{0}} ,
   /* Taula 1 */
   {{1,1} , {1,0}} ,
   /* Taula 2 */
   {{1,2,1} , {3,1,1} , {3,2,0}} ,
   /* Taula 3 */
   {{3,2,1} , {1,1,1} , {3,2,0}} ,
   /* Taula 4 */
   {{0}} ,
   /* Taula 5 */
   {{1,2,6,5} , {3,1,4,4} , {7,5,7,1} , {6,1,1,0}} ,
   /* Taula 6 */
   {{7,3,5,1} , {6,2,3,2} , {5,4,4,1} , {3,3,2,0}} ,
   /* Taula 7 */
   {{1,2,10,19,16,10} , {3,3,7,10,5,3} , {11,4,13,17,8,4} ,
   {12,11,18,15,11,2} , {7,6,9,14,3,1} , {6,4,5,3,2,0}} ,
   /* Taula 8 */
   {{3,4,6,18,12,5} , {5,1,2,16,9,3} , {7,3,5,14,7,3} ,
   {19,17,15,13,10,4} , {13,5,8,11,5,1} , {12,4,4,1,1,0}} ,
   /* Taula 9 */
   {{7,5,9,14,15,7} , {6,4,5,5,6,7} , {7,6,8,8,8,5} , {15,6,9,10,5,1} ,
   {11,7,9,6,4,1} , {14,4,6,2,6,0}} ,
   /* Taula 10 */
   {{1,2,10,23,35,30,12,17} , {3,3,8,12,18,21,12,7} , {11,9,15,21,32,40,19,6} ,
   {14,13,22,34,46,23,18,7} , {20,19,33,47,27,22,9,3} , {31,22,41,26,21,20,5,3} ,
   {14,13,10,11,16,6,5,1} , {9,8,7,8,4,4,2,0}} ,
   /* Taula 11 */
   {{3,4,9,24,34,33,21,15} , {5,3,4,10,32,17,11,10} , {11,7,13,18,30,31,20,5} ,
   {25,11,19,59,27,18,12,5} , {35,33,31,58,30,16,7,5} , {28,26,32,19,17,15,8,14} ,
   {14,12,9,13,14,9,4,1} , {11,4,6,6,6,3,2,0}} ,
   /* Taula 12 */
   {{9,5,16,33,41,39,38,26} , {7,5,6,9,23,16,26,11} , {17,7,11,14,21,30,10,7} ,
   {17,10,15,12,18,28,14,5} , {32,13,22,19,18,16,9,5} , {40,17,31,29,17,13,4,2} ,
   {27,12,11,15,10,7,4,1} , {27,12,8,12,6,3,1,0}} ,
   /* Taula 13 */
```

```
{{1,5,14,21,34,51,46,71,42,52,68,52,67,44,43,19} ,
{3,4,12,19,31,26,44,33,31,24,32,24,31,35,22,14} ,
{15,13,23,36,59,49,77,65,29,40,30,40,27,33,42,16} ,
{22,20,37,61,56,79,73,64,43,76,56,37,26,31,25,14} ,
{35,16,60,57,97,75,114,91,54,73,55,41,48,53,23,24} ,
{58,27,50,96,76,70,93,84,77,58,79,29,74,49,41,19} ,
{47,45,78,74,115,94,90,79,69,83,71,50,59,38,36,15} ,
{72,34,56,95,92,85,91,90,86,73,77,65,51,44,43,42} ,
{43,20,30,44,55,78,72,87,78,61,46,54,37,30,20,16} ,
{53,25,41,37,44,59,54,81,66,76,57,54,37,18,39,11},
{35,33,31,57,42,82,72,80,47,58,55,21,22,26,38,22} ,
{53,25,23,38,70,60,51,36,55,26,34,23,27,14,9,7} ,
{34,32,28,39,49,75,30,52,48,40,52,28,18,17,9,5} ,
{45,21,34,64,56,50,49,45,31,19,12,15,10,7,6,3} ,
{48,23,20,39,36,35,53,21,16,23,13,10,6,1,4,2} ,
{16,15,17,27,25,20,29,11,17,12,16,8,1,1,0,1}} ,
/* Taula 14 */
{{0}} ,
/* Taula 15 */
{{7,12,18,53,47,76,124,108,89,123,108,119,107,81,122,63} ,
{13,5,16,27,46,36,29,51,42,70,52,83,65,41,59,36} ,
{19,17,5,24,41,34,59,48,40,64,50,78,62,80,56,33} ,
{29,28,25,43,39,63,55,93,76,59,93,72,54,75,50,29} ,
{52,22,42,40,67,57,95,79,72,57,89,69,49,66,46,27} ,
{77,37,35,66,58,52,91,74,62,48,79,63,90,62,40,38} ,
{125,32,60,56,50,92,78,65,55,87,71,51,73,51,70,30} ,
{109,53,49,94,88,75,66,122,91,73,56,42,64,44,21,25} ,
{90,43,41,77,73,63,56,92,77,66,47,67,48,53,36,20} ,
{71,34,67,60,58,49,88,76,67,106,71,54,38,39,23,15} ,
{109,53,51,47,90,82,58,57,48,72,57,41,23,27,62,9} ,
{22,42,40,37,70,64,52,43,70,55,42,25,29,18,11,11} ,
{118,68,30,55,50,46,74,65,49,39,24,16,22,13,14,7} ,
{91,44,39,38,34,63,52,45,31,52,28,19,14,8,9,3} ,
{123,60,58,53,47,43,32,22,37,24,17,12,15,10,2,1} ,
{71,37,34,30,28,20,17,26,21,16,10,5,8,6,2,0}} ,
/* Taula 16 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 17 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
```

```
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 18 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 19 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 20 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
```

```
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 21 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 22 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 23 */
{{1,5,14,44,74,63,110,93,172,119,138,242,225,195,376,17} ,
{3,4,12,20,35,62,53,47,83,75,68,119,201,107,207,9} ,
{15,13,23,38,67,58,103,90,33,72,127,117,110,209,206,16} ,
{45,21,39,69,64,114,99,87,158,140,252,212,199,131,109,26} ,
{75,36,68,65,115,101,179,164,155,264,246,226,395,382,362,9} ,
{66,30,59,56,102,185,173,265,142,253,232,400,388,378,445,16} ,
{111,54,52,100,184,178,160,133,257,244,228,217,385,366,715,10} ,
{98,48,91,88,165,157,148,261,248,407,397,372,380,889,884,8} ,
{85,84,81,159,156,143,260,248,427,401,392,383,727,713,708,7} ,
{156,76,73,141,131,256,245,426,406,394,384,735,359,710,352,11} ,
{139,129,67,125,247,233,229,219,393,743,737,720,885,882,439,4} ,
{243,120,118,115,227,223,396,746,742,736,721,712,706,223,436,6} ,
{202,224,222,218,216,389,386,381,364,888,443,707,440,437,1728,4} ,
{747,211,210,208,370,379,734,723,714,1735,883,879,878,3459,865,2} ,
{377,369,102,187,726,722,358,711,709,866,1734,871,3458,870,434,0} ,
{12,10,7,11,10,17,11,9,13,12,10,7,5,3,1,3}} ,
/* Taula 24 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
```

```
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 25 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 26 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 27 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
```

```
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 28 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 29 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 30 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
{147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
{263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
{249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
{435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
{427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
{335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
{668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
{652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
{648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
{620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
{1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
{43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}} ,
/* Taula 31 */
{{15,13,46,80,146,262,248,434,426,669,685,681,653,517,1032,88} ,
{14,12,21,38,71,130,122,216,209,198,327,345,319,297,269,42} ,
{47,22,41,74,68,128,120,221,207,194,182,340,315,295,541,18} ,
{81,39,75,70,134,125,116,220,204,190,178,325,311,294,271,16} ,
```

```
  {147,72,69,135,127,118,112,210,200,188,352,323,306,285,540,14} ,
  {263,66,129,126,119,114,214,202,192,180,341,317,301,281,262,12} ,
  {249,123,121,117,113,215,206,195,185,347,330,308,291,272,520,10} ,
  {435,115,111,109,211,203,196,187,353,332,313,298,283,275,381,17} ,
  {427,212,208,205,201,193,186,177,169,320,303,286,268,514,377,16} ,
  {335,199,197,191,189,181,174,333,321,305,289,275,521,379,371,11} ,
  {668,184,183,179,175,344,331,314,304,290,277,530,383,373,366,10} ,
  {652,340,171,168,164,287,309,299,287,276,263,513,375,368,362,6} ,
  {648,322,316,312,307,302,292,284,279,261,512,376,370,364,359,4} ,
  {620,300,296,294,288,282,273,266,515,380,374,369,365,361,357,2} ,
  {1033,280,278,276,267,264,259,382,378,372,367,363,360,358,356,0} ,
  {43,20,19,17,15,13,11,9,7,6,4,7,5,3,1,3}}
  };



BYTE hlen[32][16][16] = {
  /* Taula 0 */
  {{0}} ,
  /* Taula 1 */
  {{1,3} , {2,3}} ,
  /* Taula 2 */
  {{1,3,6} , {3,3,5} , {5,5,6}},
  /* Taula 3 */
  {{2,2,6} , {3,2,5} , {5,5,6}},
  /* Taula 4 */
  {{0}},
  /* Taula 5 */
  {{1,3,6,7} , {3,3,6,7} , {6,6,7,8} , {7,6,7,8}},
  /* Taula 6 */
  {{3,3,5,7} , {3,2,4,5} , {4,4,5,6} , {6,5,6,7}},
  /* Taula 7 */
  {{1,3,6,8,8,9} , {3,4,6,7,7,8} , {6,5,7,8,8,9} , {7,7,8,9,9,9} ,
  {7,7,8,7,7,10} , {8,8,9,10,10,10}} ,
  /* Taula 8 */
  {{2,3,6,8,8,9} , {3,2,4,8,8,8} , {6,4,6,8,8,9} , {8,8,8,9,9,10} ,
  {8,7,8,9,10,10} , {9,8,9,9,11,11}} ,
  /* Taula 9*/
  {{3,3,5,6,8,9} , {3,3,4,5,6,8} , {4,4,5,6,7,8} , {6,5,6,7,7,8} ,
  {7,6,7,7,8,9} , {8,7,8,8,9,9}} ,
  /* Taula 10 */
  {{1,3,6,8,9,9,9,10} , {3,4,6,7,8,9,8,8} , {6,6,7,8,9,10,9,9} ,
  {7,7,8,9,10,10,9,10} , {8,8,9,10,10,10,10,10} , {9,9,10,10,11,11,10,11} ,
  {8,8,9,10,10,10,11,11} , {9,8,9,10,10,11,11,11}} ,
  /* Taula 11*/
  {{2,3,5,7,8,9,8,9} , {3,3,4,6,8,8,7,8} , {5,5,6,7,8,9,8,8} ,
  {7,6,7,9,8,10,8,9} , {8,8,8,9,9,10,9,10} , {8,8,9,10,10,11,10,11} ,
  {8,7,7,8,9,10,10,10} , {8,7,8,9,10,10,10,10}} ,
  /* Taula 12 */
  {{4,3,5,7,8,9,9,9} , {3,3,4,5,7,7,8,8} , {5,4,5,6,7,8,7,8} ,
  {6,5,6,6,7,8,8,8} , {7,6,7,7,8,8,8,9} , {8,7,8,8,8,9,8,9} ,
  {8,7,7,8,8,9,9,10} , {9,8,8,9,9,9,9,10}} ,
  /* Taula 13 */
  {{1,4,6,7,8,9,9,10,9,10,11,11,12,12,13,13} ,
  {3,4,6,7,8,8,9,9,9,9,10,10,11,12,12,12} ,
  {6,6,7,8,9,9,10,10,9,10,10,11,11,12,13,13} ,
  {7,7,8,9,9,10,10,10,10,11,11,11,11,12,13,13} ,
  {8,7,9,9,10,10,11,11,10,11,11,12,12,13,13,14} ,
  {9,8,9,10,10,10,11,11,11,11,12,11,13,13,14,14} ,
```

```
{9,9,10,10,11,11,11,11,11,12,12,12,13,13,14,14} ,
{10,9,10,11,11,11,12,12,12,12,13,13,13,14,16,16} ,
{9,8,9,10,10,11,11,12,12,12,12,13,13,14,15,15} ,
{10,9,10,10,11,11,11,13,12,13,13,14,14,14,16,15} ,
{10,10,10,11,11,12,12,13,12,13,14,13,14,15,16,17} ,
{11,10,10,11,12,12,12,12,13,13,13,14,15,15,15,16} ,
{11,11,11,12,12,13,12,13,14,14,15,15,15,16,16,16} ,
{12,11,12,13,13,13,14,14,14,14,14,15,16,15,16,16} ,
{13,12,12,13,13,13,15,14,14,17,15,15,15,17,16,16} ,
{12,12,13,14,14,14,15,14,15,15,16,16,19,18,19,16}} ,
/* Taula 14 */
{{0}} ,
/* Taula 15 */
{{3,4,5,7,7,8,9,9,9,10,10,11,11,11,12,13} ,
{4,3,5,6,7,7,8,8,8,9,9,10,10,10,11,11} ,
{5,5,5,6,7,7,8,8,8,9,9,10,10,11,11,11} ,
{6,6,6,7,7,8,8,9,9,9,10,10,10,11,11,11} ,
{7,6,7,7,8,8,9,9,9,9,10,10,10,11,11,11} ,
{8,7,7,8,8,8,9,9,9,9,10,10,11,11,11,12} ,
{9,7,8,8,8,9,9,9,9,10,10,10,11,11,12,12} ,
{9,8,8,9,9,9,9,10,10,10,10,10,11,11,11,12} ,
{9,8,8,9,9,9,9,10,10,10,10,11,11,12,12,12} ,
{9,8,9,9,9,9,10,10,10,11,11,11,11,12,12,12} ,
{10,9,9,9,10,10,10,10,10,11,11,11,11,12,13,12} ,
{10,9,9,9,10,10,10,10,11,11,11,11,12,12,12,13} ,
{11,10,9,10,10,10,11,11,11,11,11,11,12,12,13,13} ,
{11,10,10,10,10,11,11,11,11,12,12,12,12,12,13,13} ,
{12,11,11,11,11,11,11,11,12,12,12,12,13,13,12,13} ,
{12,11,11,11,11,11,11,12,12,12,12,12,13,13,13,13}} ,
/* Taula 16 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 17 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
```

```
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 18 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 19 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 20 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 21 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
```

```
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 22 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 23 */
{{1,4,6,8,9,9,10,10,11,11,11,12,12,12,13,9} ,
{3,4,6,7,8,9,9,9,10,10,10,11,12,11,12,8} ,
{6,6,7,8,9,9,10,10,11,10,11,11,11,12,12,9} ,
{8,7,8,9,9,10,10,10,11,11,12,12,12,13,13,10} ,
{9,8,9,9,10,10,11,11,11,12,12,12,13,13,13,9} ,
{9,8,9,9,10,11,11,12,11,12,12,13,13,13,14,10} ,
{10,9,9,10,11,11,11,11,12,12,12,12,13,13,14,10} ,
{10,9,10,10,11,11,11,12,12,13,13,13,13,15,15,10} ,
{10,10,10,11,11,11,12,12,13,13,13,13,14,14,14,10} ,
{11,10,10,11,11,12,12,13,13,13,13,14,13,14,13,11} ,
{11,11,10,11,12,12,12,12,13,14,14,14,15,15,14,10} ,
{12,11,11,11,12,12,13,14,14,14,14,14,14,14,13,14} ,
{11,12,12,12,12,13,13,13,13,15,14,14,14,14,16,11} ,
{14,12,12,12,13,13,14,14,14,16,15,15,15,17,15,11} ,
{13,13,11,12,14,14,13,14,14,15,16,15,17,15,14,11} ,
{9,8,8,9,9,10,10,10,11,11,11,11,11,11,11,8}} ,
/* Taula 24 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,10,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
```

```
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 25 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,10,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 26 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 27 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 28 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
```

```
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,10,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 29 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,10,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 30 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,10,10,10,10,10,11,11,11,8} ,
{11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
{11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,8} ,
{12,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,8} ,
{8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}} ,
/* Taula 31 */
{{4,4,6,7,8,9,9,10,10,11,11,11,11,11,12,9} ,
{4,4,5,6,7,8,8,9,9,9,10,10,10,10,10,8} ,
{6,5,6,7,7,8,8,9,9,9,9,10,10,10,11,7} ,
{7,6,7,7,8,8,8,9,9,9,9,10,10,10,10,7} ,
{8,7,7,8,8,8,8,9,9,9,10,10,10,10,11,7} ,
{9,7,8,8,8,8,9,9,9,9,10,10,10,10,10,7} ,
{9,8,8,8,8,9,9,9,9,10,10,10,10,10,11,7} ,
{10,8,8,8,9,9,9,9,10,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,9,10,10,10,10,11,11,8} ,
{10,9,9,9,9,9,9,9,10,10,10,10,10,11,11,11,8} ,
```

```
  {11,9,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
  {11,10,9,9,9,10,10,10,10,10,10,11,11,11,11,8} ,
  {11,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,8} ,
  {11,10,10,10,10,10,10,10,10,11,11,11,11,11,11,8} ,
  {12,10,10,10,10,10,10,11,11,11,11,11,11,11,11,8} ,
  {8,7,7,7,7,7,7,7,7,7,8,8,8,8,4}}
  };

BYTE hqcod[2][16] = {
  /* Taula per quadruples A (0) */
  {1,5,4,5,6,5,4,4,7,3,6,0,7,2,3,1} ,
  /* Taula per quadruples B (1) */
  {15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0}
  };

BYTE hqlen[2][16] = {
  /* Taula per quadruples A (0) */
  {1,4,4,5,4,6,5,6,4,5,5,6,5,6,6,6} ,
  /* Taula per quadruples B (1) */
  {4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
  };
```

7.6.2. *Defs.h*

```
#include <stdio.h>

#define BYTE  unsigned char
#define BOOL  unsigned char
#define DBYTE unsigned int

BYTE buffer[9][1440]; // Buffer on anirem emmagatzemant l'arxiu MP3. És un array
                      // de 9 buffers ja que en el cas teòric en que ens trobem
                      // en MPEG 1 i bitrate=32Kbps, sampling=48KHz, tindrem 768
                      // bits per frame, on 304 bit (32 bit header, 16 bit error
                      // check, 256 bit stereo side information) són per la part
                      // fixa; llavors, estàn disponibles 464 bits per la part
                      // dinàmica, en la que el punter s'aplicarà (el punter
                      // main_data_end pot apuntar enrera 4088 bits com a màxim,
                      // per tant pot apuntar enrera més de 8 frames).
                      // Cada buffer és de 1440 bytes ja que és la mida màxima a
                      // la que pot arribar un frame (MPEG 1: bitrate=320Kbps,
                      // Sampling=32KHz, o MPEG 2: bitrate=160Kbps, Sampling=16KHz).

BYTE buf_actius;      // Ens indicarà el nombre de buffers que hem omplert.
DBYTE buf_pointer[9]; // Punter que ens indicarà l'última posició ocupada del buffer

DBYTE buf_actual[9];  // Punter que ens indicarà l'última posició que hem tractat
BYTE buf_restant[9];  // Ens indicarà el nombre de bits de buffer[buf_actiu][buf_actual]
                      // que encara no hem tractat.

typedef struct
{
```

```
  DBYTE main_data_end;
  BYTE private_bits;
  BYTE scfsi[4][2];
  DBYTE part2_3_length[2][2];
  DBYTE big_values[2][2];
  DBYTE global_gain[2][2];
  DBYTE scalefac_compress[2][2];
  BYTE window_switching_flag[2][2];
  BYTE block_type[2][2];
  BYTE mixed_block_flag[2][2];
  BYTE table_select[2][2][3];
  DBYTE subblock_gain[2][2][3];
  BYTE region0_count[2][2];
  BYTE region1_count[2][2];
  BYTE preflag[2][2];
  BYTE scalefac_scale[2][2];
  BYTE count1table_select[2][2];
} SideInfoDataStructure;

typedef struct
{
  float msec;                 // Durada de la trama
  int n_channels;
  float values[2][2][576]; // Valors de les components freqüencials
} FrameInformation;
```