An advanced interpreter for musical language

Noms: Yeray Bosoms Blasco i Ferran Noguera Vall
Assignatura: Compiladors
Data: 2/6/2017

1- Introducció

El nostre projecte, per l'assignatura de compiladors, s'ha basat en la creació d'un intèrpret musical. Aquest llegeix i interpreta un arxiu pla amb partitura (escrita d'una forma que explicitarem al llarg d'aquesta memòria) i, en acabar, ho reprodueix.

És un intèrpret bastant complet ja que s'han implementat la gran majoria de funcionalitats que es troben comunament en una partitura, malgrat li n'han falten algunes, com podrien ser adorns y articulacions de notes, però aquestes són minoritàries i sense elles igualment es poden escriure la gran majoria de cançons, així doncs hem decidit deixar-les com una possible ampliació futura del intèrpret.

El intèrpret s'ha basat en una llibreria per Java (actualment encara en desenvolupament) anomenada Jfugue, la qual ens ha facilitat l'implementació de les funcionalitats que es troben habitualment en una partitura. Entrarem en més detall sobre aquest sistema software al llarg d'aquesta memòria.

També hem deixat un conjunt de jocs de prova, de diferents dificultats pel intèrpret, els quals reprodueixen un seguit de cançons conegudes i n'explicitem les diferents funcionalitats de les quals disposa el intèrpret.

El nostre objectiu principal quan vam rebre aquesta pràctica i el perquè la vam escollir va ser poder realitzar la interpretació d'un dels jocs de prova esmentats: "binks' sake" ja que als dos ens agrada la sèrie on apareix i la cançó en general. Al obrir la partitura vam veure que tenia una gran quantitat de funcionalitats i ens vam posar com a repte aplicar-les totes en el nostre intèrpret, aquest és el motiu pel que hem decidir aplicar aquests símbols musicals i no d'altres (entenent que aplicar-los tots era impossible, basant-se en el temps del que disposàvem per fer la pràctica, però plantejable en un futur).

2. Principals Funcionalitats del Llenguatge

Hem implementat un seguit de funcionalitats: la gran majoria d'elles referents a la interpretació i execució d'una partitura, una funcionalitat per la realització de bucles for i les funcionalitats per incrementar, decrèixer, sumar, restar, multiplicar o dividir una mateixa variable i guardar-ho a memòria ("++" "- -" "+=" "-=" "*=" "/="). S'han mantingut totes les funcionalitats inicials que ens venien donades amb l'Asl i no s'han modificat, per tant no les comentarem més al llarg de la memòria.

A continuació explicitarem les funcionalitats del nostre llenguatge:

For

Aquesta funcionalitat servirà per realitzar iteracions. Té el mateix format que el for trobat a C++ (assignació de variable, comparació i operació de canvi sobre la variable assignada). El programa no sortirà de la iteració fins que la comparació no es compleixi.

Increment/Decrement/Suma/Resta/Producte/Divisió d'una mateixa variable

Aquesta funcionalitat permet la realització d'aquestes instruccions per una mateixa variable sense haver-ho d'escriure convencionalment i per tant més llarg (Per exemple: x = x+2 -> x+=2). S'ha creat aquesta funcionalitat perquè la programació amb el llenguatge sigui més simple i estalviar-se instruccions massa llargues quan d'aquesta manera queda un codi més fàcil i entendible.

Funcions

Cada programa haurà de contenir la funció principal "func main() { (instruccions aquí) }" però apart d'aquesta en podrem inicialitzar més dins del codi. Aquestes podran rebre diferents paràmetres (per referència o per valor), aquests podran ser del tipus Enter, Booleà, String o Nota musical. Aquestes funcions podran ser void o retornar algun valor (amb "return"), no serà necessari especificar-ho doncs totes s'inicialitzaran amb el mateix format que el main, però canviant aquest pel nom de la funció en qüestió i posant els paràmetres necessaris dins dels parèntesis i separats per punt i comes.

Creació d'una nota musical

La creació més bàsica d'una nota musical contindrà: nota, octava i duració (en aquest ordre i separant aquests dos últims amb un punt, un exemple seria: "Sol4.n", on Sol indicaria la nota, 4 l'octava i n la duració.

Les notes són “Do”, “Re”, “Mi”, “Fa”, “Sol”, “La”, “Si” i “Quiet”. Aquestes representaran la tonalitat musical del tipus Nota Musical.

Les octaves es comprendran des del 1 fins al 8, tenint per defecte el 3. Aquestes tindran el mateix efecte en la nota per la definició d’octava.

La duració de la nota anirà representat al final, seguit d’un punt. Aquestes podran ser, ordenades de major a menor tonalitat: “r”, “b”, “n”, “c”, “sc”, “f”, “sf” (rodona, blanca, negre, surera, semicorxera, fusa i semifusa, respectivament).

Alteració Musical

Les alteracions que podrà tenir seran: “Bemoll”, “Sostingut” i “Becaire” representades com: “$”, “#”, “¬”, respectivament. Aquestes tindran el mateix efecte en la nota que tenen normalment en qualsevol partitura musical.

Aquestes es situaran al inici de la nota que es vulgui alterar, per exemple: “#Fa3.n”.

Operacions per notes musicals

Les notes musicals disposaran de 4 operacions bàsiques, aquestes que seran: Raise, Fall, Half i Twice. Raise elevarà mig to i Fall baixarà mig to, Half disminuirà la duració a la meitat i Twice la doblarà. Aquestes operacions es situaran davant de la nota musical que es vulgui que rebi aquesta modificació. Es podran combinar les 4 operacions entre si, aplicant-se d’esquerra a dreta, infinitament.

Octava per defecte

Hem implementat l’opció de no situar cap octava en la nota musical, si es fa això el intèrpret entendrà que et refereixes a l’octava 3. S’ha afegit aquesta funcionalitat degut a que la gran majoria de partitures usen molt aquesta octava i això facilitarà al programador la generació de notes musicals, estalviant escriure de més.

Acumulació de duracions

Al igual que l’octava, també s’ha implementat l’opció de que no s’escrigui cap duració al final de la nota musical, si es realitza això el intèrpret entendrà que la duració de la nota musical serà la mateixa que la de la nota musical anterior, clarament aquesta funcionalitat no funcionarà si la nota musical en qüestió és la primera de la partitura. Aquesta funcionalitat s’ha afegit pel mateix motiu que la funcionalitat “octava per defecte”.

Punt

El punt afegirà a la nota musical la meitat de la duració que tenia. Aquesta funcionalitat es representarà amb un * al final de la duració, per exemple: “La5.b*”.

Aquesta funcionalitat no permetrà l’acumulació per duracions.

Multinotas

Hi haurà l’opció de que diferents notes musicals (amb diferent tonalitat i/o octava) sonin a la vegada, amb la mateixa duració. Per escriure una multinota s’haurà d’escriure les notes que es vulgui tocar entre parèntesis, separades per comes i sense la duració, ja que aquesta s’especificarà al final del parèntesis amb un punt i la duració corresponent.

Aquesta funcionalitat perdrà les funcionalitats de “operacions per notes musicals” i tampoc es podrà passar una multinota com a paràmetre ni entrada/sortida, però mantindrà les demés especificades.

Triet

El triet permetrà que es toquin les notes musicals contingudes dins seu a ⅔ de la seva duració real. Per escriure un triet s’haurà de fer entre “[“ “]”, dins hi haurà notes musicals amb tonalitat, octava i duració. No es separarà per comes.

Pont

Lligarà dos o més notes musicals fent que sonin com una, per tant sumant la duració en la que es tocaran. Per representar-ho ho farem amb un ~ al final de la nota que vulguem que faci pont amb l’anterior, la següent nota que no tingui aquest símbol s’entendrà que és la que tanca el pont, per exemple: “La4.n~ La4.b~ La4.r” això generarà un pont del primer La al últim i la duració seria la d’una negra més una blanca més una rodona.

ID

Una ID estarà formada per un String, Enter, Booleà o Nota Musical. Tenint en compte que aquest últim no disposarà de les funcionalitats: “Pont, Multinota, Triet i operacions per notes musicals” però absolutament totes les demés.

Beat

El beat marcarà el total de duracions de notes que pot haver-hi per cada fragment de la partitura. Aquest estarà format per dos expressions numèriques o ID's, separades per dos punts. Aquest tindrà el mateix efecte en la partitura que el que té normalment en qualsevol partitura musical, essent el número de davant dels dos punts el que es situa a dalt i el de darrera el de baix.

Speed

El speed es refereix al tempo de la partitura, aquest marca la quantitat de pulsacions per minut que es tocaran. Està format per una duració musical (rodona, negra, blanca, etc.) i una expressió numèrica o ID, unit per dos punts. Aquest tindrà el mateix efecte en la partitura que el que té normalment en qualsevol.

Voice

La partitura podrà sonar amb diferents instruments (per defecte el piano), però s'haurà d'especificar sempre Voice abans de col·locar les notes, per defecte serà amb Voice sense cap paraula seguida, si, en canvi, s'especifica Voice seguit d'un instrument haurà de sonar la partitura amb l'instrument escrit.

Els instruments són: "Piano", "Flauta", "Shot", "Trompa", "Organo", "Viola", "Saxo", "Guitarra", "Percusion", "Xilofono".

Chorus

Aquesta funcionalitat permetrà que diferents (o el mateix instrument) soni a la vegada (a cor). Com a mínim un chorus estarà format per dos instrument i màxim setze. El chorus s'indicarà escrivint: "Chorus" seguit de "{ }" on dins hi aniran el total de voice's amb els instruments corresponents.

Creació d'una partitura

Una partitura simple, amb tots els elements obligats necessitarà començar amb un Beat, seguit d'un Speed. A partir d'aqui perquè s'executi hi haurà d'haver, com a mínim, una Voice o un Chorus i com a màxim qualsevol combinació d'aquests dos.

Seguit d'una Voice hi haurà d'haver un conjunt de fragments (amb les notes marcades pel el Beat corresponent), cadascun d'ells separat per "|", començant i acabant també amb elles.

Transport

Aquest mou el número de tons que li posis i afecta a totes les notes d'una partitura, menys les que estiguin amb una alteració Becaire. Es situa just després del Speed i va seguit d'una expressió numèrica o ID.

Armadura

L'armadura és un conjunt d'alteracions que s'aplica a tota la partitura, menys les que estiguin amb una alteració Becaire. Aquesta es situa just després del transport (si hi és, sinó l'Speed). Es declara posant una llista dins de "{ }" amb la nota, alteració i octava a la que afecten (s'ha de tenir en compte que si s'aplica una alteració a una nota musical (per exemple La) afectarà al La de totes les octaves de la partitura i s'han d'introduir manualment.

Ritornelo

Hi haurà bucles de fragments en la partitura "ritornelo". S'haurà d'escriure una expressió numèrica, o una variable, el qual significarà la quantitat de vegades que vols que es repeteixi el fragment, seguit de "z||:". A continuació s'escriurà el fragment de partitura, de la mateixa manera descrita en l'anterior paràgraf, i finalment ":||". Durant la repetició del fragment, existirà una variable interna "Time" que indicarà el número de repeticions que porta i que es podrà fer servir als condicionals interns de la partitura. Aquesta començarà essent zero en la primera iteració i augmentarà d'un en un per cadascuna de les iteracions.

Condicionals de partitura

Existirà la possibilitat de declarar condicionals enmig de la partitura, per fer-ho s'haurà de seguir la mateixa estructura que en els condicionals normals, amb un "if" condició (tenint en compte que pot haver-hi la variable interna Time anteriorment nombrada) i un seguit de notes i, llavors, pot haver-hi un else amb un altre grup de notes diferent. Les notes situades dins el if o l'else tenen la restricció que han de ser com a màxim un compàs i escrites sense "|".

Gestió d'error de la mètrica

Existirà un control d'errors que permetrà al programador saber si ha comés algun error en la mètrica, indicant el fragment on l'ha comés i la partitura on l'ha comés. Hem decidit implementar aquesta gestió d'errors perquè és molt fàcil equivocar-se en la duració d'una nota alhora de passar d'una partitura al nostre llenguatge i aquesta era una manera d'assegurar que es cometien menys errors.

3. Descripció del llenguatge i modificacions respecte l'ASL

Variables i tipus

Aquest és un llenguatge de programació sense tipus, qualsevol variable pot tenir qualsevol tipus i aquest poden canviar dinàmicament. Com en molts llenguatges interpretats, les variables no es declaren, es creen quan se'ls assigna un valor, la comprovació de tipus es fa dinàmicament en temps d'execució.

El llenguatge no té variables globals, totes les variables són locals a la funció on són creades, l'únic mecanisme de comunicació que hi ha entre funcions es el pas de paràmetres.
Les funcions son també sense tipus i poden retornar qualsevol tipus de valor (o res) depenen de la seva execució.
El punt i coma s'utilitza per separar instruccions individuals, així com també per acabar una partitura

Paramètres

Els paràmetres també son sense tipus, hi ha dos mecanismes per passar paràmetres, per valor y per referència, es distingeixen afegint el símbol & davant el nom del paràmetre a la declaració per referència.
Al cridar la funció no es necessari especificar si es para per valor o per referència, el mecanisme es inferit per la declaració de paràmetres al definir la funció, es poden passar expressions com a paràmetre per valor, nomes variables poden ser passades per referència

Expressions

La comprovació de tipus es fa quan s'avaluen expressions, les aritmètiques requereixen operands aritmètics, les booleanes booleans i les musicals requereixen notes musicals.

El intèrpret

Ha estat dissenyat amb ANTLRv3 i Java, per instal·lar-ho cal fer un 'make' al directori arrel del llenguatge. Per executar l'interpret es fa cridant l'archiu Asl de la carpeta bin de la forma Asl <archiu>, amb la possibilitat de diverses opcions que poden ser llistades amb Asl -help.

Organització del intèrpret

Els arxius font de intèrpret estan organitzats en 3 packs:

-Asl-> conte la classe Asl, que conte el main del programa i crida al parser i al intèrpret.

-parser-> conte la gramàtica i el creador de ast, que en generen mitjançant ANTLR

-interp-> conte el intèrpret de l'AST en diversos arxius:

Interp.java: conte el nucli del interpret, recorrent l'AST i executant les instruccions.

Data.java: conte la classe que representa els valors de les dades (Integer Boolean i Nota) i executa operacions en elles

Stack.java: Implementa la memòria del intèrpret amb una pila de blocks d'activació que contenen parells de strings(noms de variables) i data(valors)

AslTree.java: conte una subclasse de l'AST que amplia la informació inclosa en cada node de l'AST.

AslTreeAdaptor.java: conte una subclasse requerida per ANTLR per tenir accés a l'AST ampliat.

Interp.java

Principalment té 2 funcions publiques:

Interp(AslTree T) és el constructor del intèrpret, prepara l'AST per la seva interpretació. També crea una taula que mapeja els noms de funció amb el nodes AST on estan definides. El paràmetre T es l'arrel de l'AST

void Run(): executa el programa interpretant l'AST.

En aquest arxiu també hi ha funcions essencials que constitueixen el nucli del intèrpret.

Data executeFunction(String funcname, AslTree args): Aquesta funció executa una funció donats el nom i el node AST on hi son els paràmetres passats a la crida, la funció també implementa el mecanisme de pas de paràmetres.

Data executeInstruction(AslTree t): Aquesta funció executa una instrucció individual, en el cas de que la funció retorni un valor, aquesta funció el retorna, en cas contrari retornara un null.

Data evaluateExpression(AslTree t): Aquesta funció retorna el valor produït per l'avaluació de l'expressió representada per un l'AST.
Durant l'execució del programa l'intèrpret realitza diverses revisions de semàntica (accés a variables no definides, operacions amb tipus incompatibles, etc) que llençaran excepcions en temps l'execució, l'intèrpret també manté un seguiment del numero de línia on succeeix l'excepció per un millor debuggeix.

Per la part de interpretació de partitures, el programa es basa en les tres funcions prèvies i 4 funcions mes:

int metrica(AslTree t): és una funció que revisa que la partitura apuntada pel node t compleixi el beat especificat, llençant una excepció quan no el compleixi. Té un seguiment de número de fragment i número de partitura per un millor debuggeix.

void cargaTrasnport(AslTree t) i void cargaArmor(AslTree t): carreguen respectivament el valor de Transport a aplicar a les notes de la partitura actual y el valor de l'armadura. Els carrega en una matriu global que es restableix en cada partitura.

void tract_voces(AslTree t): és la funció que avalua la partitura Voice a Voice (Chorus inclosos) i genera la llista de notes a reproduir amb tots els components de la partitura.

Data.java

Aquesta classe representa un objecte guardat a memòria. Proporciona mètodes per obtenir el tipus i el valor de data, així com també algunes característiques addicionals pel tipus Note. Proporciona a mes mètodes per realitzar operacions aritmètiques, lògiques i alguns mètodes per revisions semàntiques i de temps d'execució.

Stack.java

Aquesta classe representa la memòria de la maquina virtual que interpreta el codi. el mecanisme de pas de paràmetres funciona tenint un bloc d'activació associat a cada funció que emmagatzema totes les variables de dins scope de la funció. quan

es crida una funció es crea un nou bloc d'activació, i quan s'acaba la crida es destrueix. D'aquesta forma la memòria esta organitzada com una pila de blocs d'activació.
Cada bloc conte un set de <noms, valors> amb totes les variables associades als seus valors corresponents, o a una referència a una entrada d'un altre bloc d'activació en el cas de les variables passades per referència. Les entrades d'un bloc d'activació son creades en temps d'execució cada cop que es defineix una nova variable

4- Software Addicional Usat

Com software addicional usarem la llibreria de java JFugue. És una llibreria open source, encara en vies de desenvolupament que utilitza les capacitats MIDI de Java per per generar musica de forma programàtica.

Aquesta ens permet reproduir música especificant instruments, notes, acords, temps i demés funcionalitats especificades anteriorment d'una manera relativament senzilla.

Aquest també ens permet generar, llegir, interpretar i reproduir arxius en format MIDI, però ens hem centrat únicament en la part de creació i interpretació de partitures. Malgrat això, una ampliació del intèrpret "simple" que es podria realitzar seria aquesta, degut a que ja disposem d'un software que ens permetria fer-ho.

5- Jocs de Prova

Hem generat un seguit de jocs de prova, els quals s'han basat en transformar diferents partitures que hem trobat per Internet de cançons populars i que tinguessin demostracions de funcionalitats, del nostres llenguatge, interessants.

Hem intentat que les partitures anessin augmentant en dificultat tenint en compte que l'última de totes conté totes les funcionalitats del llenguatge i és una cançó llarga i d'alta dificultat.

Himne del alegria

L'arxiu de nom "himnodelalegria.asl" conté aquesta coneguda cançó a saxòfon. Com podem observar en usa diferents funcionalitats del llenguatge, com poden ser: ritornelo, armadures, transportes i condicionals de partitura.

El ritornelo s'expressa des del fragment 1 fins el 5 i el condicional de partitura en el fragment 4 i 5 (usant la variable interna Time). També es pot comprovar que hi ha multitud de notes en els fragments del 1 al 3 sense octava ni duració que apliquen l'explicat en les funcionalitats del llenguatge en quan a creació de notes.

Addicionalment també hem creat l'arxiu de nom "himnodelalegriaSW.asl", el qual és la mateixa cancó, però aquest cop interpretada a més d'un instrument (Guitarra, Saxo, Trompa, Shot, Percusion i Organo).

D'aquest arxiu en destacarem:

- L'ús de diferents Voice (amb diferents instruments) per una mateixa declaració de partitura.
- L'ús de funcions (passant i sense passar paràmetres de qualsevol tipus).
- La declaració d'ID's per notes diferents: *"nota = Si.n", "notaa = Sol.n*"*.
- L'us del for i l'increment d'una mateixa variable (línia 3: "*++aux*").
- Operacions per notes musicals amb notes (a notes amb i sense alteracions): *"Half La.b"* fragment 2 funció repe i *"Raise $Mi"* fragment 3 de l'anterior funció.

Moonriver

L'arxiu de nom: "moonriver.asl" conté aquesta popular cançó interpretada a flauta. D'aquest joc de proves en destaquem: l'ús de ponts, en la funció repeat fragment 5: *"Re.b*~"*, com es pot observar també podem veure que funcionen els punts amb els ponts i sense, com podria ser el cas de la nota: *"Si.n*"*, fragment 1 de la funció repeat.

The office (opening)

L'arxiu de nom "office.asl" conté l'opening de la serie "The office" tocat a piano. Com podem veure per interpretar cada mà del piano ho farem amb "Chorus", on hi haurà dos voice's.

D'aquest joc de proves en destacarem:

- L'aparició de diferents alteracions com poden ser el sostingut: "#Fa.c" fragment 2.
- Per primera vegada que apareixen multinotas (amb o sense alteracions i ponts): *"(Si3,Re4).c"* fragment 1 de la primera voice. *"(#Fa4,Re4,Si3).c"* fragment 14 de la primera voice i els quatre últims fragments de la segona voice són multinotes amb ponts: *"(Sol2,Sol3).r~ | (Sol2,Sol3).r~ | (Sol2,Sol3).r~ | (Sol2,Sol3).r"*

Canon de Pachelbel

Aquest joc de proves l'hem inclòs per les funcionalitats que aporta, sinó per demostrar que, sense programar-ho implícitament, es permet crear funcionalitats addicionals. En aquest cas podem comprovar que jugant amb els silencis entre chorus hem pogut crear un Canon entre Trompa, Flauta i Xilòfon.

Binks' Sake

L'arxiu de nom: "bnsb2.asl" conté una cançó pirata interpretada a piano. D'aquest joc de proves en destaquem:

- L'aparició de triets nomes amb notes normals com al fragment 2 :*[Mi4.n Fa4.c],*
- Triets amb multinotes (amb alteracions incloses) com al fragment 1*[(¬La4, #Fa4).n Si3.c]*
- Triets amb pont : com al fragment 2*[(Sol4, Mi4).n Mi4.c~]*
- Notes amb alteració múltiple com per exemple *"$$$Do1.n"*, del fragment 1 de la segona Voice del Chorus
- L'aparició de becaires dins de triet amb pont entre les seves components com al fragment 33 *[La3.f ¬La3.sc*~ La3.c La3.c]*
- La durada de la cançó i la seva complexitat, per tenir una armadura complexa (12 notes afectades), moltes alteracions, una gran quantitat de multinotes i triets de tota mena, condicionals i ritornelos.

6- Possibles extensions

Hi ha una gran varietat de possibles extensions no implementades en aquest intèrpret, de les quals una gran majories d'elles les hem anat comentant al llarg de la memòria, a continuació en destacarem algunes (per ordre de simplicitat de codi), tenint en comptem com està:

1. **Addició de més instruments**: S'hauria de modificar lleument el Asl.g i el Interp.java per tal d'afegir nous instruments, els quals no seria gens complicat doncs en la llibreria de Jfugue n'hi ha encara una gran varietat que no hem considerat per afegir.

2. **Guardar Multinotas:** Seria modificar els arxius Data.java i Interp.java, perquè admetés un vector de notes. Aquesta modificació seria lleujerament complexa doncs implicaria modificar gran part del Data.java.

3. **Afegir qualsevol símbol musical:** Nosaltres només hem afegit els símbols més comuns i necessaris per escriure partitures, però hi ha una gran quantitat d'ells que no hem inclòs. Aquest canvi augmentaria de dificultat segons quants més símbols volguessis posar, però en qualsevol cas en cadascun d'ells implicaria modificar tan el intèrpret Interp.java com el parser Asl.g (i depenent del símbol el Data.java).

4. **Addició de tractament d'arxius MIDI:** Aquesta extensió seria bastant complexa, doncs implicaria números os canvis en arxius com Interp.java i Asl.g, doncs el llenguatge no està preparat actualment per rebre cap arxiu MIDI, però el software addicional que estem usant: "jfugue" conté diferents funcions que podrien fer aquesta extensió més lleugera i plantejable en una futura extensió d'aquest intèrpret.