DOI 10.37882/2223–2966.2022.06–2.34

БИБЛИОТЕКА SDL КАК ВСПОМОГАТЕЛЬНЫЙ ИНСТРУМЕНТ АЛГОРИТМИЧЕСКОГО ДИЗАЙНА ПРИ СОЗДАНИИ КОМПОЗИЦИЙ И СИНТЕЗИРОВАННЫХ ЗВУКОВЫХ КОМБИНАЦИЙ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ NYQUIST

Таран Василий Васильевич
К.культурологии, заведующий, Лаборатория компьютерного дизайна и прикладной информатики «SPLASH»; ФГБУН «Всероссийский институт научной и технической информации РАН»
allscience@lenta.ru

Аннотация. В статье проанализирован материал, касающийся применения библиотеки SDL в Nyquist-программировании. Основным целеполаганием авторского научного анализа является качественное ускорение процессов трансляции традиционной нотации счёта на исходный код языка Nyquist. Раскрыты потенциальные функции библиотеки SDL, способствующие совершенствованию практик алгоритмического дизайна с использованием языка Nyquist. Установлено, что процедуры счёта библиотеки SDL основываются на создании табличных списков, внутри которых указываются математические параметры и некоторые символьные данные, способствующие быстрой и качественной калькуляции в условиях перевода нотации счёта в микросценарии и процедурный код языка Nyquist. На фрагментарных примерах кода показана его структура и отображены правила изложения логических конструкций, формирующих процессы счёта по инструкциям библиотеки SDL. Разобраны и уточнены основные функции библиотеки SDL, влияющие на формирование событийного аппарата при Nyquist-программировании. Обосновано, что применение инструментария библиотеки SDL значительно упрощает и улучшает прикладные процедуры по переводу нотации счёта в процедурный код языка Nyquist.

Ключевые слова: Nyquist-программирование, библиотека SDL, процедурный код, библиотеки языков программирования, алгоритмический дизайн композиций, синтезированные звуковые комбинации, процедуры счёта, табличные списки, нотация счёта, символьно-числовые данные, калькуляция массивов аудиоданных.

SDL LIBRARY AS AN AUXILARY ALGORITHMIC DESIGN TOOL WHEN CREATING COMPOSITIONS AND SYNTHETIC AUDIO COMBINATIONS BY USING NYQUIST PROGRAMMING LANGUAGE

V. Taran

Summary. The matter dealing with the implementation of the SDL library in Nyquist programming is investigated in the article below. The main purpose of the scientific analysis suggested by the author is the qualitative acceleration of the conventional translation of score notation into basic Nyquist language code. The potential abilities of the SDL library contributing the maintenance of algorithmic design practices by using the Nyquist language are revealed. It is established that the calculating procedures of the SDL library are based on the generation of tabular lists containing mathematical parameters and some symbolic data contributing fast and high-quality calculation when translating the score notation into microscenaries and the procedural code of the Nyquist language. Individual code samples demonstrate its structure and rules for the presentation of logical constructions defining calculating process according to the SDL library manuals. The main functions of the SDL library affecting the formation of the event-driven mechanism in Nyquist programming are analyzed and clarified. It is proved that the use of the SDL library tools significantly simplifies and improves the application procedures for translating score notation into procedural code of the Nyquist language.

Keywords: Nyquist programming, SDL library, procedural code, the libraries of the programming languages, algorithmic design of compositions, synthetic audio combinations, calculating procedures, tabular lists, score notation, symbolic and numerical data, calculation of audio data array.

Библиотеки языков программирования являются важным структурным элементом в сфере разработки программно-технических решений, направленных на совершенствование инженерных практик в различных отраслях человеческой деятельности. Инженерные практики в области компьютерной обработки звука постоянно обогащаются, прежде всего, за счёт инновационных процессов, происходящих в одной из *главных наук современности* — компьютерных наук. Не секрет, что в настоящее время, практически ни одна инженерная практика не обходится без вовлечения в неё компьютерных технологий. Компьютерные технологии в сфере обработки аудиоданных дают возможность аудиоинженеру раскрыть творческий потенциал при процедурах синтеза звука, его мастеринга и ремастеринга, а также прочих прикладных процедур, многим из которых посвящён ряд авторских статей [1,2,3,4,5,6,7,8,9].

Программирование — это технический инструментарий, позволяющий достичь желаемых результатов в каждой из предметных областей инженерных практик. Качественное программирование — вещь сложная, а его процесс требует колоссальной внимательности и порой принятия нестандартных решений. Чтобы сосредоточиться исключительно на *инженерных процедурах* и не растрачивать своё время на *рутинные практики программирования*, можно использовать *специализированные библиотеки*, целью которых является обеспечение непрерывности процессов программирования в условиях реализации сложноалгоритмизированных проектов.

Специальные языки программирования (отраслевые языки) обладают богатым спектром возможностей для реализации креативных концепций проектировщика. Язык программирования Nyquist (далее по тексту — язык Nyquist), относится именно к таким языкам. Язык Nyquist позволяет производить сложные манипуляции над звуковыми данными, упрощает процедуры формирования аудиоданных, решает проблему точной обработки звука при проектировании новых композиций, предоставляет возможность произведения синтеза звуковых комбинаций, превращая набор сэмплов в совершенно новые звуки.

Язык Nyquist поддерживает исполнение команд на синтаксисе SAL[1]. Это обстоятельство позволяет исполнять функции счёта в Nyquist на SAL. Как и другие языки подобного толка, Nyquist нуждается во вспомогательных средствах программирования, которые могут существенно облегчить рутинные процедурные моменты в режиме формирования программного кода. Для этого в Nyquist зарезервирован определённый набор библиотек, предназначенных для упрощения мини операций, связанных с написанием кода. Одной из таких библиотек является SDL [10,11].

Библиотека SDL (**S**core **D**escription **L**ibrary) является библиотекой описания счёта и предназначена для облегчения перевода традиционной нотации счёта на исходный код языка Nyquist[2], она также успешно обеспечивает точный контроль над параметрами производительности и синтеза. Библиотека спроектирована Педро Моралесом (Pedro J. Morales)[3]. Цель разработки данной библиотеки заключается в попытке *облегчения транскрипции музыки*, от *традиционной нотации* — до *синтеза*, на языке Nyquist. Строго говоря, композиции компьютерной музыки формально задаются с помощью предопределённого языка программирования. Благодаря тому, что язык Nyquist является расширением языка LISP, он может использоваться одинаково хорошо как для *синтеза аудио сэмплов*, так и для *алгоритмической композиции*[12]. Связь Nyquist с основным LISP, а также с некоторыми его ветками (XLISP, Common Lisp), расши-

[1] **Прим. автора.** SAL (**S**imple **A**lgorithmic **L**anguage) — алгоритмический язык программирования специального назначения, спроектированный в целях обеспечения удобства трансляции алгоритмических моделей в машинный код. Язык может быть полезен электронным композиторам, инженерам по звуку, которые хотят реализовать свои замыслы на компьютерных (программно-аппаратных) платформах. Является Lisp-ориентированным языком.

[2] **Прим. автора.** Nyquist поощряет представление музыки в виде исполняемых программ или моделей поведения. Существуют различные способы изменения поведения, включая растяжение во времени, транспонирование и т.д. Альтернативой составлению исполняемых программ является манипулирование со счётом, как с редактируемыми данными. У каждого подхода есть свои сильные и слабые стороны. Мы описываем функции, предназначенные для управления счётом Xmusic, полученные с помощью score-gen. Напомним, что таким образом счета являются списками событий (например, нот), где события представляют собой трехэлементные списки формы (выражение продолжительности времени, и где выражение является стандартным вызовом функции LISP, где все параметры являются параметрами *ключевых слов*). Кроме того, первой «нотой» может быть специальное выражение «SCORE-BEGIN-END». Если оно отсутствует, партитура начинается с нуля и заканчивается последней нотой.

[3] **Прим. автора.** Хуан Педро Домингес-Моралес (Juan Pedro Dominguez-Morales) — специалист в области компьютерных наук и аудиоинженерии, получил степень бакалавра в области компьютерной инженерии, степень магистра в области компьютерной инженерии и сетей, и степень доктора философии в области компьютерной инженерии (специализируется на *нейроморфной* обработке звука и нейронных сетях с шипами) в Университете Севильи, Севилья, Испания, в 2014, 2015 и 2018 годах — соответственно. С октября 2015 по декабрь 2018 года он был аспирантом факультета архитектуры и компьютерных технологий Севильского университета, получив исследовательский грант от Министерства образования и науки Испании. С января 2019 года работает доцентом на кафедре архитектуры и компьютерных технологий. Его научные интересы включают анализ медицинских изображений, свёрточные нейронные сети, системы автоматизированной диагностики, *нейроморфную* инженерию, импульсные нейронные сети, *нейроморфные* датчики и обработку звука. В 2016 году он стал членом Европейского общества нейронных сетей и уже четыре года является членом Института инженеров электротехники и электроники — IEEE (Institute of Electrical and Electronics Engineers).

```
(setf *my-sdl-score*
'((TF 1)
(INIT-INSTR "i1" sinte) (INSTRUMENT "i1") (PWL-POINT :mm 100) (ATTR :idur 0.1)
2 (:c4 1) :d4 (:e4 2) :c4 :g4 :e4 :a4 (:g4 1) :f4 (:g4 4)
(:a4 1) :c5 :f4 :a4 :g4 :f4 :e4 :g4 :f4 :a4 :d4 :f4 :e4 :d4 :c4 :d4 :e4 :f4 (:g4 2)
:a3 :c4 :d4 :f4
:g3 :b3 :c4 :e4 (:f3 1) :e4 :d4 :c4 :g3 :d4 :c4 :b3 (ATTR :idur 1) (LABEL :t1)
(:c4 4)))
```


Программный код № 1*. Базовый пример счёта SDL.
*Иллюстрации к статье являются неотъемлемой частью авторского права данной рукописи, их воспроизведение в публичных научных источниках (в том числе средствах массовой информации) допускается с письменного разрешения редакции и уведомления автора. Фрагменты программного кода, изложенные в иллюстрациях, принадлежат их законным владельцам: Педро Моралесу (Pedro J. Morales) и Луису Родригесу (Luis Rodr´ıguez).

ряет возможности традиционного программирования за счёт введения интеллектуальных функций[13,14]. Помимо этого, язык Nyquist можно эффективно использовать для воспроизведения музыки, которая описывается как последовательность нот (партитура в традиционной нотной записи). Но в этом случае необходимо *указать каждый параметр ноты*, что затрудняет комфортное сочинение музыки[15,16,17,18].

Счёт SDL *во многом упрощает данные процедуры* и представляет собой табличный список, в котором указаны инструктивные примечания[1], а также атрибуты продолжительности звука, высота его тона и другие важные аргументы, связанные с параметрами синхронизации и синтеза.

Здесь TF означает фактор времени. Все значения длительности будут умножены на этот коэффициент (значение по умолчанию равно 1). INIT-INSTR показывает инструмент, который будет использоваться при синтезе. Первый аргумент «i1» — это название инструмента в счёте SDL. Второй аргумент — это имя функции Nyquist, которое определяет инструмент. Базовый перечень функций счёта на языке Nyquist перечислен в таблице 1. Эта функция должна быть определена независимо от счёта с помощью выражения (defun sinte …). Команда ИНСТРУМЕНТ (INSTRUMENT) способствует тому, что все ноты с этого момента синтезируются инструментом «i1» до тех пор, пока не будет указан новый инструмент. PWL-POINT описывает линейную функцию по частям, для того, чтобы задать *переменные* во времени параметры. К примеру: mm принимает значение 100, когда инструкция, называемая ATTR, задаёт значение постоянного параметра[2]. Значение 2 определяет двухтактную паузу. Можно рассмотреть несколько различных типов времени:

- Время счёта, измеряемое в тактах.
- Физическое время, измеряемое в секундах.
- Четверть — имеет продолжительность 4 удара.

Физическое время вычисляется в соответствии со временем счёта, темпом и Временным Фактором TF. (: c4 1) представляет собой ноту, заданную шагом C4 и длительностью в 1 такт (т.е. под номером шестнадцатая). Указываются только высота и продолжительность. Шаг может быть задан с помощью альтернативного синтаксиса. Длительность может быть любым выражением LISP.

Остальные атрибуты, необходимые для синтеза, предоставляются инструкциями ATTR и PWL-POINT. «d4» — это нота высоты тона D4 и унаследованная длительность 1 такта. Остатки не изменяют продолжительность по умолчанию. Изменения продолжительности за-

[1] **Прим. автора.** Под инструктивными примечаниями понимаются примечания, содержащие набор правил и предписаний для своевременного их выполнения на определённых участках программного кода. Это может быть зарезервированный для конкретных целей сектор таблицы или ячейка списка, которым должна соответствовать определённая процедура.

[2] **Прим. автора.** Значение параметра не изменяется до тех пор, пока не будет достигнута новая инструкция ATTR. Например, параметр: idur принимает значение 0,1 в каждой заметке до тех пор, пока новое значение не будет указано новым значением ATTR.

Таблица 1. Перечень основных функций, предназначенных для счёта при составлении программного кода на языке Nyquist.

Функции счёта Nyquist (свойства и события)			Процедурные предписания
Диалект SAL		Диалект LISP	
1	score-sorted(score)	(score-sorted score)	Проверяет, отсортирован ли счёт.
2	score-sort(score [, copy-flag])	(score-sort score [copy-flag])	Сортирует ноты в партитуре *от начала*. Если разделитель копирования равен нулю, это «разрушительная» операция, которая должна выполняться только в том случае, если список оценок верхнего уровня является свежей копией, которая не используется другими переменными. Разделитель копирования предназначен только для использования во внутренней системе. Для следующих операций предполагается, что счета отсортированы, и все операции возвращают отсортированный счёт.
3	score-shift(score, offset, from-index: i, to-index: j, from-time: x, to-time: y)	(score-shift score offset: from-index i: to-index j: from-time x: to-time y)	Добавляет постоянное смещение ко времени начала набора нот в партитуре. По умолчанию все ноты изменяются, но их диапазон может быть ограничен параметрами ключевого слова. Время начала оценки при необходимости уменьшается до минимального времени любого события, перенесённого на более раннее время (с отрицательным смещением), а время окончания оценки увеличивается при необходимости до максимального времени окончания любого события, перенесённого на более позднее время. Если все смещенные события остаются в диапазоне от начала до конца счёта, время начала и окончания не изменяется. Исходный счёт не изменяется, а возвращается новый счёт.
4	score-stretch(score, factor, dur: dur-flag, time: time-flag, from-index: i, to-index: j, from-time: x, to-time: y)	(score-stretch score factor: dur dur-flag: time time-flag: from-index i: to-index j: from-time x: to-time y)	Растягивает время и продолжительность нот по алгоритму. Разделитель dur по умолчанию не равен нулю, но если разделитель dur равен нулю, первоначальная длительность сохраняется и растягивается только время. Аналогично, разделитель времени по умолчанию не равен нулю, но если равен нулю, исходное время сохраняется, и растягиваются только продолжительности. Если и разделитель dur, и разделитель времени равны нулю, счёт не изменяется. Если указан диапазон нот, время масштабируется в пределах этого диапазона, а ноты после диапазона сдвигаются так, чтобы растянутая область не создавала «дыру» или не перекрывалась бы последующими заметками. Если диапазон начинается или заканчивается временем (через «from-time:» и «to-time:»), растяжение времени происходит течение указанного временного интервала независимо от того, присутствуют ли какие-либо ноты или где они начинаются. Другими словами, «остатки» растягиваются вместе с нотами. Исходный счёт не изменяется, а возвращается новый счёт.
5	score-transpose(score, keyword, amount, from-index: i, to-index: j, from-time: x, to-time: y)	(score-transpose score keyword amount: from-index i: to-index j: from-time x: to-time y)	Для каждой ноты в партитуре и в любом указанном диапазоне, если есть параметр, соответствующий ключевому слову и значение параметра выражено числом, увеличивает значение параметра на определенную величину. Например, чтобы транспонировать вверх на целый шаг, напишите ((score-transpose 2: pitch score). Исходный счёт не изменяется, а возвращается новый счёт. Если ключевое слово «: pitch» и соответствующее значение параметра является списком, каждый элемент списка увеличивается на определенную величину. Этот особый случай соответствует соглашению timed-seq, в котором события оценки со списками для атрибута «: pitch» расширяются в «аккорды», путём создания экземпляра события для каждого элемента (шага) в списке (аккорде).

Таблица 1 (продолжение). Перечень основных функций, предназначенных для счёта при составлении программного кода на языке Nyquist.

Функции счёта Nyquist (свойства и события)			Процедурные предписания
Диалект SAL		Диалект LISP	
6	score-scale(score, keyword, amount, from-index: i, to-index: j, from-time: x, to-time: y)	(score-scale score keyword amount: from-index i: to-index j: from-time x: to-time y)	Для каждой ноты в партитуре и в любом указанном диапазоне, если есть ключевое слово, соответствующее ключевому слову, и значение параметра равно числу, умножает значение параметра на определённую величину. Исходный счёт не изменяется, а возвращается новый счёт.
7	score-sustain(score, factor, from-index: i, to-index: j, from-time: x, to-time: y)	(score-sustain score factor: from-index i: to-index j: from-time x: to-time y)	Для каждой ноты в партитуре и в любом указанном диапазоне умножает продолжительность (фактор растяжения) на определённую величину. Это может быть использовано для того, чтобы ноты звучали более выраженным легато или стаккато, и не изменяет время их начала. Исходный счёт не изменяется, а возвращается новый счёт.
8	score-voice(score, replacement-list, from-index: i, to-index: j, from-time: x, to-time: y)	(score-voice score replacement-list: from-index i: to-index j: from-time x: to-time y)	Для каждой ноты в партитуре и в любом указанном диапазоне, заменяет имя (функции) поведения с помощью replacement-list, который имеет формат: ((old1 new1) (old2 new2) …), где old — показывает текущее имя поведения, а new — является заменой. Если old обозначается как *, это может соответствовать любому типу данных. Например, чтобы заменить my-note-1 тромбоном, а my-note-2 рожком, используйте score-voice(score, {{my-note-1 trombone} {my-note-2 horn}}). Чтобы заменить все инструменты на фортепиано, используйте score-voice(score, {{* piano}}). Исходный счёт не изменяется, а возвращается новый счёт.
9	score-merge(score1, score2, …)	(score-merge score1 score2 …)	Создаёт новый счёт, содержащий все примечания к параметрам, которые являются счетами. Полученные ноты сохраняют свое первоначальное время и продолжительность. Время начала объединенной оценки — это минимальное время начала параметров, а время окончания объединенной оценки — максимальное время окончания параметров. Исходные счета не изменяются, а возвращается новый счёт.
10	score-append(score1, score2, …)	(score-append score1 score2 …)	Создаёт новый счёт, содержащий все примечания к параметрам, которые являются счетами. Время начала первого счета остается неизменным. Время начала каждого другого счета совпадает со временем окончания предыдущего счёта; таким образом, счета «сращиваются» последовательно. Исходные счета не изменяются, а возвращается новый счёт.

Таблица 1 (продолжение). Перечень основных функций, предназначенных для счёта при составлении программного кода на языке Nyquist.

Функции счёта Nyquist (свойства и события)			Процедурные предписания
Диалект SAL		Диалект LISP	
11	score-select(score, predicate, from-index: i, to-index: j, from-time: x, to-time: y, reject: flag)	(score-select score predicate: from-index i: to-index j: from-time x: to-time y: reject flag)	Выбирает (или отклоняет) ноты, чтобы сформировать новую партитуру. Ноты выбираются, если они попадают в заданные диапазоны индекса и времени, и удовлетворяют предикату *(Предикат в программировании — выражение, использующее одну или более величину с результатом логического типа), функции из трёх параметров, которая применяется к времени начала, продолжительности и нотному выражению. В качестве альтернативы предикатом может быть t, указывающее, что должны быть выбраны все ноты в диапазоне. Выбранные ноты, вместе с существующими маркерами начала и конца партитуры, объединяются для формирования новой партитуры. В качестве альтернативы, если параметр «reject:» не равен нулю, невыбранные ноты образуют новый счёт (другими словами, выбранные ноты отклоняются или удаляются для формирования нового счёта).* Исходный счёт не изменяются, а возвращается новый счёт.
12	score-get-begin(score)	(score-get-begin score)	Возвращает начальное время счёта.
13	score-set-end(score, time)	(score-set-end score time)	Конечное время начала с маркера счёта SCORE-BEGIN-END устанавливается на время. Исходный счёт не изменяются, а возвращается новый счёт.
14	score-get-end(score)	(score-get-end score)	Возвращает время окончания счёта.
15	score-must-have-begin-end(score)	(score-must-have-begin-end score)	Если счёт не имеет времени начала и окончания, создаёт счёт с выражением SCORE-BEGIN-END и возвращает его. Если у счёта уже есть время начала и окончания, просто возвращает счёт. Исходный счёт не изменяется.
16	score-filter-length	(score-filter-length score cutoff)	Удаляет ноты, которые выходят за рамки времени отсечения. Это похоже на score-select, но события здесь удаляются, когда их номинальное время окончания (время начала плюс продолжительность) превышает отсечку, в то время как параметр «to-time:» сравнивается со временем начала ноты. Исходный счёт не изменяется, а возвращается новый счёт.
17	score-repeat(score, n)	(score-repeat score n)	Создаёт последовательность из n копий партитуры. Каждая копия смещается так, чтобы время её начала совпадало со временем окончания предыдущей копии, как в score-append. Исходный счёт не изменяется, а возвращается новый счёт.
18	score-stretch-to-length(score, length)	(score-stretch-to-length score length)	Растягивает счёт так, чтобы время окончания счёта было временем начала счёта, плюс его длиной. Исходный счёт не изменяется, а возвращается новый счёт.
19	score-filter-overlap(score)	(score-filter-overlap score)	Удаляет перекрывающиеся ноты (на основе времени начала и продолжительности заметки), отдавая приоритет порядку расположения в списке нот (который также является временным порядком). Исходный счёт не изменяется, а возвращается новый счёт.

Таблица 1 (продолжение). Перечень основных функций, предназначенных для счёта при составлении программного кода на языке Nyquist.

Функции счёта Nyquist (свойства и события)			Процедурные предписания
Диалект SAL		Диалект LISP	
20	score-print(score, [lines])	(score-print score [lines])	Печатает партитуру с одной нотой в строке. Возвращает ноль. Если задано количество строк (необязательно ФИКСИРОВАННОЕ число), выводит максимум из строк (но минимально не менее 3-х). Формат — первый параметр lines-2 события оценки, строка «…» и последнее событие оценки.
21	score-play(score)	(score-play score)	Воспроизводит партитуры с помощью timed-seq для преобразования партитуры в звук и воспроизведение для демонстрации звука.
22	score-adjacent-events(score, function, from-index: i, to-index: j, from-time: x, to-time: y)	(score-adjacent-events score function: from-index i: to-index j: from-time x: to-time y)	Вызывает (функции A B C), где A, B и C — последовательные ноты в партитуре. Полученный результат заменяет B. Если результат равен нулю, B удаляется, и следующий вызов будет (функция A C — D) и т.д. Первый вызов — (функция ноль A B), а последний — (функция Y Z ноль). Если в партитуре есть только одна нота, вызывается (функция ноль, A — ноль). Вызовы функций не выполняются, если нота находится за пределами указанного диапазона. Эта функция позволяет настраивать ноты и их параметры в соответствии с их непосредственным контекстом. Исходный счёт не изменяется, а возвращается новый счёт.
23	score-apply(score, function, from-index: i, to-index: j, from-time: x, to-time: y)	(score-apply score function: from-index i: to-index j: from-time x: to-time y)	Заменяет каждую ноту в партитуре результатом (выражение dur функции времени) (в LISP) или функцией (время, dur, выражение) (в SAL), где время, dur и выражение — это время, продолжительность и выражение ноты. Если указан диапазон, заменяются только ноты в диапазоне. Исходный счёт не изменяется, а возвращается новый счёт.
24	score-indexof(score, function, from-index: i, to-index: j, from-time: x, to-time: y)	(score-indexof score function: from-index i: to-index j: from-time x: to-time y)	Возвращает индекс (*позицию*) *первого* события оценки (в диапазоне), для которого применяется функция, посредством (выражения dur/ функция времени) с последующим возвратом реального значения.
25	score-last-indexof(score, function, from-index: i, to-index: j, from-time: x, to-time: y)	(score-last-indexof score function: from-index i: to-index j: from-time x: to-time y)	Возвращает индекс (*позицию*) *последнего* события оценки (в диапазоне), для которого применяется функция, посредством (выражения dur/ функция времени) с последующим возвратом реального значения.
26	score-randomize-start(score, amt, from-index: i, to-index: j, from-time: x, to-time: y)	(score-randomize-start score amt: from-index i: to-index j: from-time x: to-time y)	Изменяет время начала нот на случайную величину до плюс или минус amt. Исходный счёт не изменяется, а возвращается новый счёт.

даются явно атрибутом примечания. Также есть функции для изменения времени начала нот, времени растяжения и длительности, растяжения только длительности, добавления смещения к параметру ключевого слова, масштабирования параметра ключевого слова и других манипуляций. Предусмотрены функции для извлечения диапазонов нот, нот, соответствующих критериям, и для объединения счётов. Большинство из этих функций (подробно перечисленных ниже) имеют общий набор параметров ключевых слов, которые при необходимости ограничивают диапазон, в котором выполняется преобразование. Параметры «from-index:» и «to-index:» показывают индекс первой ноты и индекс последней ноты, которую необходимо изменить. Если эти числа отрицательные, они смещены от конца партитуры, например, «-1» обозначает последнюю ноту партитуры. «from-time:» и «to-time:»[1] — идентифицируют диапазон времени начала нот, на которые повлияет манипуляция. Изменяются только ноты, время которых больше или равно времени «from-time» и строго меньше времени «to-time». Если указаны как индексы, так и временные диапазоны — выбираются только заметки, удовлетворяющие обоим ограничениям.

LABEL — задаёт метку времени, предназначенную для синхронизации разделов счёта. LABEL связана со временем подсчёта. Совпадение ссылок на LABEL зависит

[1] **Прим. автора.** В синтаксисе LISP двоеточия предшествуют ключевому слову, поэтому используйте «: from-index», «: to-index», «: from-time» и «: to-time».

```
(load ''sdl'')
(sdl->score *my-sdl-score*)

=> ((0.3 0.15 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
(0.45 0.15 (SINTE :PITCH 62 :MM 100 :IDUR 0.1))
(0.6 0.3 (SINTE :PITCH 64 :MM 100 :IDUR 0.1))
(0.9 0.3 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
(1.2 0.3 (SINTE :PITCH 67 :MM 100 :IDUR 0.1))
...
)
```


Программный код № 2. Обработка оценки SDL.

```
(sdl:normalize-score-duration(sdl->score *my-sdl-score*))

=> ((0.3 1 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
(0.45 1(SINTE :PITCH 62 :MM 100 :IDUR 0.1))
(0.6 1 (SINTE :PITCH 64 :MM 100 :IDUR 0.1))
(0.9 1 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
(1.2 1 (SINTE :PITCH 67 :MM 100 :IDUR 0.1))
...
)
```


Программный код № 3. Установка масштабирующих множителей растяжения на 1 по sdl: normalize-score-duration.

```
((0.3 0.15 (SINTE :PITCH 60 :MM 100 :IDUR 0.1))
```

Рис. 1

от карты темпа оценки. Таким образом, пользователь несёт ответственность за контроль над этой проблемой. Библиотека счёта может производить событийную оценку, указываемых математических параметров. Оценка SDL обрабатывается функцией sdl->score.

Как можно заметить, в результате получается оценка Xmusic[1]. Каждое событие Xmusic score состоит из трёх элементов. Первый из них указывает время начала за-

[1] **Прим. автора.** Несколько библиотек Найквиста предлагают поддержку алгоритмической композиции. Xmusic — это специальная библиотека для генерации последовательностей и шаблонов данных. В Xmusic включены:

- ✓ объекты шаблонов, используемые для генерации интересных последовательностей значений параметров;

Music работает на Common Lisp и Scheme, но не на XLISP (базовом языке для Nyquist). Библиотеки Xmusic в Nyquist предлагают интересное подмножество инструментов в Common Music. Одной из важных особенностей Xmusic является то, что он интегрирован со всеми функциями синтеза Nyquist, поэтому вы можете использовать шаблоны и партитуры Xmusic для управления мелкими деталями синтеза звука.

* Рик Таубе (Rick Taube) является доцентом в Школе музыки при Университете Иллинойса в Урбане (США). Специализируется на компьютерной обработке звука, алгоритмической композиции, теории классической композиции.

** Common Music (CM) — это система создания музыки в реальном времени, реализованная на JUCE/C++ и Scheme. Она генерирует музыкальный вывод через MIDI, OSC, CLM, FAMOUS и CSOUND. Её пользовательское приложение называется GRACE (Графическая среда алгоритмической композиции в реальном времени). Фактически является объектно-ориентированной средой создания музыки, распространяемой по лицензии GPL. Common Music — создаёт звук, преобразуя высокоуровневое представление музыкальной структуры в различные протоколы управления для синтеза и отображения аудиоматериала.

```
(setf *my-time-labels* (sdl->timelabels *my-sdl-score*))

(setf *voz-2*
'((TF 1)
(INIT-INSTR "i1" sinte) (INSTRUMENT "i1") (PWL-POINT :mm 100) (ATTR :idur 0.1)
(AT-LABEL :t1)
2 (:g4 1) :a4 (:b4 2) :g4))

(sdl->score *voz-2* *my-time-labels*)

=> ((9.9 0.15 (SINTE :PITCH 67 :MM 100 :IDUR 0.1))
(10.05 0.15 (SINTE :PITCH 69 :MM 100 :IDUR 0.1))
(10.2 0.3 (SINTE :PITCH 71 :MM 100 :IDUR 0.1))
(10.5 0.3 (SINTE :PITCH 67 :MM 100 :IDUR 0.1)))
```


Программный код № 4. Процедуры синхронизации. Пример отправки временных ссылок в качестве аргумента в функцию sdl->score.

```
(play (timed-seq (sdl:normalize-score-duration (sdl->score *my-sdl-score*)))
```

Рис. 2

```
(setf *my-time-labels* (sdl->timelabels *my-sdl-score*))
(print (symbol-plist *my-time-labels*))
=> (:T1 64)
```

Рис. 3

```
(setf *minimal-sco-1* '((INIT-INSTR "i1" xx) (INSTRUMENT "i1") (PWL-POINT :mm 100)
                        (:c4 2)))
(setf *minimal-sco-2* '((INIT-INSTR "i2" yy) (INSTRUMENT "i2") (ATTR :mm 100) :c4))
```

Рис. 4

метки. Второй — это фактор растяжения, а третий — вызов функции синтеза. Например, первое событие (рис. 1).

Оно начинается с 0,3 секунды; коэффициент растяжения равен 0,15, а вызов функции синтеза (синтез: шаг 60: idur 0,1). Здесь: аргумент mm —это не параметр синтеза, а элемент управления темпом.

Таким образом, реализация sinte не должна содержать никаких аргументов с этим именем. Масштабирующий множитель растяжения умножает длительность ноты на значение растяжения. Например, для вызова функции синтеза для возврата 1-секундной заметки, масштабирующий множитель растяжения равен 0,15, а общая продолжительность заметки составит 0,15. Одной из наиболее примечательных особенностей Nyquist является Поведенческая абстракция (Behavioral Abstraction), которая представляет собой основу для решения контекстно-зависимых преобразований, включая растяжение. Каждая функция синтеза имеет опре-

```
(setf *pitches* (list 60 60.0 c4 'c4 C4 'C4 cs4 df4 :c4 :c#4 :cb4 :df4
"c4" "cs4" "cb4" "c#4"))
(mapcar #'sdl:pitch-lex *pitches*)
=> (60 60 60 60 60 60 61 61 60 61 59 62 60 61 59 61)
```

Рис. 5

деленное поведение по умолчанию, которое адекватно в большинстве ситуаций. Тем не менее, в некоторых случаях это поведение по умолчанию может быть неприемлемым. Поэтому, например, воздействие на огибающую амплитуду звука может быть изменено различными значениями масштабирующего множителя растяжения и может не подходить для звука, подобного клавесину. Чтобы решить эту проблему, функцию синтеза можно определить так, чтобы время достижения максимального уровня было постоянным и не зависело от масштабирующего множителя растяжения. Для выполнения этой операции требуются обширные знания Поведенческой абстракции (Behavioral Abstraction). В качестве альтернативы *всемасштабирующего множителя,* растяжения могут быть установлены равными 1 и для указания конкретной продолжительности любого события можно использовать определенный параметр. Для этой цели используется параметр: idur.

Таким образом, продолжительность звука может быть выше или ниже, что позволяет использовать несколько режимов артикуляции от легато[1]. Параметры контроля времени не могут быть вызваны: dur, так как это имя является зарезервированным ключевым словом Xmusic. «sdl: normalize-score-duration» — используется для установки масштабирующих множителей растяжения на 1, как показано в следующем *примере*. Программный код №3.

Для удобства представления счёт можно визуализировать. Как только счёт SDL будет переведён в формат Xmusic, Nyquist behavior может быть определен с помощью функции timed-seq[2], рис. 2.

При работе со *сложными оценками* целесообразно разбивать их на более мелкие этапы. Временные ссылки должны быть установлены для синхронизации всех задействованных этапов. Эти временные привязки могут быть установлены с помощью LABEL-инструкции. Функция sdl->timelabels используется для получения списка временных ссылок. Эти данные регистрируются как *список свойств символа*, созданного по ходу выполнения. *Например* (рис. 3).

Инструкция AT-LABEL позволяет нам сопоставить временную метку, которая была ранее указана LABEL, а также другое значение времени из другого счёта. Список временных ссылок должен быть отправлен в качестве аргумента в функцию sdl->score, как показано в следующем примере:

Пользователи должны учитывать, что метки времени на самом деле являются временем счёта (измеряемого в тактах), тогда как в оценках Xmusic время указывается в секундах, которые, в свою очередь, отображаются на основе времени счёта, параметра TF и карты темпа. По этой причине *метки времени* всегда совпадают со *временем счёта*, но для достижения совпадения в *физическом времени* они должны использовать один и тот же параметр TF и карту темпа. Счёт SDL — это табличный список, содержащий примечания, записи или другие элементы, связанные с синтезом и общим процессом. В любом счёте SDL должны присутствовать следующие базовые элементы:

- Инициализация одного инструмента INIT-INSTR.
- Одно задание по измерению INSTRUMENT.
- Одно временное функциональное требование: mm с помощью PWL-POINT или ATTR.
- По крайней мере, одна нота.

Пример — рис. 4.

Функция sdl->score возвращает ошибку, если счёт не содержит этих основных данных.

Здесь была принята стандартная шкала высоты тона MIDI (60 = C4). При использовании чисел с плавающей запятой (FLONUM) не допускается изменение высоты тона. Используем стандартные символы Nyquist.

Пример:

cs4 = C4 sharp = 61.

[1] **Прим. автора.** Легато (итал. legato — связанно, плавно) в музыке — приём игры на музыкальном инструменте или в пении, связное исполнение звуков, при котором имеет место плавный переход одного звука в другой (пауза между звуками отсутствует) до стаккато*.

*Стаккато (итал. staccato — отрывисто) — музыкальный штрих, предписывающий исполнять звуки отрывисто, отделяя один от другого паузами).

[2] **Прим. автора.** Также можно использовать функцию воспроизведения партитуры для непосредственного её отображения.

```
(setf *sco3*
'((TF 1.0) ; Общий фактор времени
(INIT-INSTR ''i1'' xx) ; Преамбула(инициализация) instr. init.
(INSTRUMENT ''i1'') ; Присвоенный инструмент.
(ATTR :mm 60) ; Метроном MM=60. 1 Четверть = 1 сек.
(:c4 4) ; c4 Четверть
4 ; Оставшаяся Четверть
:d4 ; d4. Продолжительность = четверть (наследуется)
(PAU 8) ; Оставшаяся половина
(:e4 (/ 4.0 3.0)) ; e4. Треть четверти
:e4 ;
:e4 ;
(DELTA (* 2 4)) ; Оставшаяся половина
(:f4 (* 2 1.5)) ; f4 Продолжительность = 3 = пунктирная восьмая часть
(DUR 4) ; Текущая продолжительность = 4 (четверть)
:g4 ; Продолжительность g4 = четверть
(DUR (/ 4.0 3.0)) ; текущая продолжительность = треть четверти
:a4 :a4 :a4 ; тройные структурные элементы a4
))
```

Программный код № 5. Пример использования инструкций DELTA и PAU при работе с продолжительностью числового значения (действия указаны в комментариях к коду — «;»).

```
(setf *sco4*
'((TF 1.0) ; Общий фактор времени
(INIT-INSTR ''i1'' xx) ; Преамбула init. instr.
(INSTRUMENT ''i1'') ; Присвоенный  инструмент
(ATTR :mm 60) ; Метроном MM=60. 1 Четверть = 1 сек.
(:c4 4) ; четверть c4
4 ; Оставшаяся четверть
(DUR 8) ; Текущая продолжительность =8= половина
(CH :c4 :e4 :g4) ; Трёхнотный аккорд. Продолжительность = половина
```

Программный код № 6. Пример краткосрочной продолжительности по CH. Использование трёхтонного аккорда, краткосрочная продолжительность — ½.

Стандартные звуковые символы lambda Music[1] представлены символами, начинающимися с двоеточия. Острые и плоские точки обозначаются # и b.

Пример:

«: c4: C4: c#4: C#4: cb4: Cb4».

Табличные символы.

Пример: 'c4 'cs4 'c#4 'cb4.

Строковые последовательности.

Пример: "c4""cs4""c#4""cb4" (рис. 5).

Продолжительность измеряется в тактах. Четверть эквивалентна 4 тактам, половина — 8 тактам и *так далее*. Допускаются дробные продолжительности. Продолжительность может быть задана:

- Числовым значением.
- Выражением (интерпретацией) LISP.

Пример:

(/ 4.0 3.0) — представляет собой треть четверти, то есть одну восьмую каждого из трёх элементов.

[1] **Прим. автора.** Библиотека музыкальных композиций, разработанная Педро Моралесом в Common Lisp.

```
(setf *sco5*
'((TF 1.0) ; Общий фактор времени
(INIT-INSTR ''i1'' xx) ; Преамбула Init. instr.
(INSTRUMENT ''i1'') ; Присвоенный инструмент
(ATTR :mm 60) ; Метроном MM=60. 1 Четверть = 1 сек.
(:c4 4) ; Четверть c4
4 ; Оставшаяся четверть
(CH1 :c4 4) ; НАЧАЛО АККОРДА. Четверть c4
(CH1 :e4 8) ; e4 половина. ОДНОВРЕМЕННОЕ НАЧАЛО c4
(:g4 4) ; g4 четверть. ОДНОВРЕМЕННОЕ НАЧАЛО. c4, e4
(:c5 4) ; c5 четверть. Начинается, когда g4 заканчивается
; e4 продолжает играть из-за половины продолжительности
(CH1 :c4 8) ; НАЧАЛО арпеджированного аккорда
;(звуки воспроизводятся поочередно)
(DELTA 0.2) ; Увеличение времени на 2 такта
(CH1 :e4 (- 8 0.2)) ; e4 начинается на 0,2 удара после c4
; Заканчивается в то же самое время
(DELTA 0.2) ; Время увеличивается на 2 такта
(CH1 :g4 (- 8 0.2 0.2)) ; g4  другая нота арпеджио
(:c5 (- 8 0.2 0.2 0.2)) ; c5  финальная нота арпеджио
(:g3 4) ; g3 Четверть после арпеджио
)) ; (общая продожительность арпеджио = 8)
```


Программный код № 7. Использование DELTA-инструкции в качестве инструмента приращения времени, каждая нота может иметь независимые аккорды по длительности.

```
(setf *sco6*
'((TF 1.0)
(INIT-INSTR ''i1'' flute) ; Присвоить название партитуры инструмента через i1
; для функции флейты
(INSTRUMENT ''i1'') ; Текущий инструмент = i1
(ATTR :mm 60)
(:c4 4) ; Передано флейтой
4
(INIT-INSTR ''i2'' clarinet) ; Присвоить название партитуры инструмента через i2
; Для функции кларнета
:d4 ; Передано флейтой
(INSTRUMENT ''i2) ; Текущий инструмент = i2
:e4 ; Передано кларнетом
))
```


Программный код № 8. Пример сопоставления имени счётного инструмента с функцией Nyquist по INIT-INSTR.

```
(setf *sco7*
'((TF 1.0)
(INIT-INSTR ''i1'' flute) ; Карты i1 для флейты
(INSTRUMENT ''i1'') ; Текущий инструмент i1(флейта)
(:c4 4) ; mm = 100; rel = 4; затухание = 3.2
(ATTR :decay 3.2) ;
(:d4 4) ; mm = 100; rel = 4; затухание = 5.1
(ATTR :decay 5.1)
(:e4 4) ; mm = 100; rel = 4; затухание = 5.1
(PWL-POINT :rel 4.0)
(PWL-POINT :mm 100)
(:f4 4) ; mm = 100; rel = 4; затухание = 5.1
(:g4 4) ; mm = 100; rel = 5; затухание = 5.1
(:a4 4) ; mm = 100; rel = 6; затухание = 5.1
(PWL-POINT :rel 7.0)
(:b4 4) ; mm = 100; rel = 7; затухание = 5.1
(:c5 4) ; mm = 100; rel = 7; затухание = 5.1
(:d5 4) ; mm = 100; rel = 7; затухание = 5.1
(PWL-POINT :rel 7.0)
(:e5 4) ; mm = 100; rel = 7; затухание = 5.1
))
```

Программный код № 9. Пример определения параметров функций синтеза в Nyquist при счёте SDL по ATTR и PWL-POINT.

Физическое время (измеряется в секундах) вычисляется после времени счёта (измеряется в тактах), фактора времени, TF и карты темпов (задаётся параметром: mm). Фактор глобального времени задаётся инструкцией по оценке TF. Пример: (TF 2.0) умножает на 2,0 все длительности в счёте, тогда как (TF 0,5) умножает это на 0,5. Эта инструкция должна появиться в счёте только один раз. Значение TF по умолчанию равно 1,0. Значения темпов задаются параметром: mm, значение которого устанавливается с помощью инструкций ATTR или PWL-POINT. Синхронизация по примечаниям осуществляется через две возможности синхронизации нот:

1) С помощью 2-х элементного списка (высота и продолжительность).

Пример:

(: c4 4) является четвертью C4

2) Только по высоте. Продолжительность взята из последней ноты.

Продолжительность по умолчанию может быть изменена с помощью инструкции DUR. Что же касается дополнительной информации, можно отметить, что числовое значение представляет количество тактов. Инструкции DELTA или PAU[1] указывают на продолжительность с помощью числового значения или выражения LISP.

Музыкальная полифония может быть реализована в SDL путём сбора нескольких партитур. Тем не менее, заранее заданные (встроенные) инструкции SDL могут

[1] **Прим. автора.** Обе инструкции взаимозаменимы.

```
(setf *sco-8*
'((TF 1)
4 (:c4 4)
(PWL-POINT :mm 60)
:d4 :e4 :f4 :g4
(PWL-POINT :mm 60)
:e5 :d5 :c5 :b4 :c5
(PWL-POINT :mm 30)))

(score-print (sdl->score *voz-8*))

=> ((0 1 (FLUTE :PITCH 60 :MM 60))
(1 1 (FLUTE :PITCH 62 :MM 60))
(2 1 (FLUTE :PITCH 64 :MM 60))
(3 1 (FLUTE :PITCH 65 :MM 60))
(4 1 (FLUTE :PITCH 67 :MM 60))
(5 1 (FLUTE :PITCH 76 :MM 60))
(6 1.11111 (FLUTE :PITCH 74 :MM 54)) ; Pumapgango
(7.11111 1.25 (FLUTE :PITCH 72 :MM 48))
(8.36111 1.42857 (FLUTE :PITCH 71 :MM 42))
(9.78968 1.66667 (FLUTE :PITCH 72 :MM 36))
)
```

Программный код № 10. Применение PWL-POINT для постепенного изменения темпа замедления воспроизведения музыкальной дорожки. Для резкого изменения темпа музыкального произведения используем ATTR.

быть более подходящими в простых случаях. CH — производит одновременные ноты, имеющие краткосрочную продолжительность. *Пример* — прогаммный код №6.

CH1 выдает одну ноту без увеличения времени. Приращения времени могут быть указаны с помощью DELTA-инструкции. Таким образом, для каждой ноты могут быть созданы независимые аккорды длительности.

Пример — программный код № 7.

В каждой партитуре ноты должны быть привязаны к инструменту синтеза. INIT-INSTR — это инструкция, используемая для сопоставления имени счётного инструмента с функцией Nyquist, реализующей сам инструмент. Инструкция INSTRUMENT присваивает примечаниям название инструмента. Текущий инструмент можно настроить с помощью новой инструкции INSTRUMENT.

В счётах SDL параметры функций синтеза в Nyquist определяются инструкциями ATTR и PWL-POINT. У ATTR есть *два* аргумента. *Первый* — это имя параметра (начинающееся с двоеточия). *Второй* — используется только для установки значения параметра. Текущее значение параметра может быть изменено другой инструкцией ATTR. PWL-POINT ведёт себя так же, как инструкция ATTR. Единственное *различие* заключается в том, что *линейная интерполяция* выполняется между *двумя последовательными инструкциями* PWL-POINT. Не разрешается определять одно и то же значение характеристики с помощью инструкций ATTR и PWL-POINT.

ATTR можно использовать в случае резких изменений темпа, в то время как PWL-POINT (Ritardando)[1] — по-

[1] **Прим. автора.** Ritardando (ритардандо) — постепенное замедление темпа исполнения музыкального произведения.

```
; Функция Найквиста, повторяющая событие n раз

(defun sdl-repeat (n quoted-event)
(let (result)
(dotimes (i n (apply #'append result))
(push quoted-event result))))

; Макрос sdl-repeat, вызванный из счета
; Последовательность :f4 :g4 повторяется 4 раза

(setf *score*
'((TF 1.0)
(INIT-INSTR "i1" flute2)(INSTRUMENT "i1")(ATTR :mm 60)
(:e4 4) (MAC sdl-repeat 4 (:f4 :g4))))

(sdl->score *score*))

=> ((0 1 (FLUTE2 :PITCH 64 :MM 60))
(1 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4 повторите 4 раза f4 g4
(2 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
(3 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4
(4 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
(5 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4
(6 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
(7 1 (FLUTE2 :PITCH 65 :MM 60)) ; f4
(8 1 (FLUTE2 :PITCH 67 :MM 60)) ; g4
)
```


Программный код № 11. Пример работы с макросами. Применение инструкции MAC для повторяющихся структур.

```
; Поля, управляемые логистическим уравнением
; Логистическая кривая*
; Комментарий*:  Таран Василий Васильевич, кандидат культурологии

(setf *logistica* 0.3)
(setf *k* 3.97)

(defun logistica ()
(setf *logistica* (* *k* (- 1.0 *logistica*) *logistica*)))

(defun call-logistica (n dur pitch-min pitch-interval)
(let (result)
(dotimes (i n (reverse result))
(push (list ( pitch-min (round (* (logistica) pitch-interval))) dur)
result))))

(setf *score2*
'((TF 1.0)
(INIT-INSTR "i1" flute2)(INSTRUMENT "i1")(ATTR :mm 60)
(:e4 4) (MAC call-logistica 5 4 30 15)))

(score-print (sdl->score *score2*))

=> ((0 1 (FLUTE2 :PITCH 64 :MM 60))
(1 1 (FLUTE2 :PITCH 43 :MM 60))
(2 1 (FLUTE2 :PITCH 38 :MM 60))
(3 1 (FLUTE2 :PITCH 45 :MM 60))
(4 1 (FLUTE2 :PITCH 31 :MM 60))
(5 1 (FLUTE2 :PITCH 34 :MM 60))
)
```


Программный код № 12. Пример использования управления значениями параметров с помощью функции LISP для автоматизации работы с примечаниями.

```
(load "xm")

(setf *my-durations* (make-heap (list 4 6 8)))

(defun heap-durations (inicio pitch)
(let ((dur (next *my-durations*)))
(list inicio dur (list 'sinte-fun :pitch pitch :mm 60 :idur dur))))

(setf *score3*
'((TF 1.0)
(INIT-INSTR "i1" flute2)(INSTRUMENT "i1")(ATTR :mm 60)
(:e4 4)
(FUN #'heap-durations 1 :c4)
(FUN #'heap-durations 2 :d4)
(FUN #'heap-durations 3 :e4)
(FUN #'heap-durations 4 :c4)
(FUN #'heap-durations 7 :d4)))

(sdl->score *score3*)

=> ((0 1 (FLUTE2 :PITCH 64 :MM 60))
(0.25 1.5 (SINTE-FUN :PITCH :C4 :MM 60 :IDUR 6))
(0.5 1 (SINTE-FUN :PITCH :D4 :MM 60 :IDUR 4))
(0.75 2 (SINTE-FUN :PITCH :E4 :MM 60 :IDUR 8))
(1 1.5 (SINTE-FUN :PITCH :C4 :MM 60 :IDUR 6))
(1.75 1 (SINTE-FUN :PITCH :D4 :MM 60 :IDUR 4))
)
```


Программный код № 13. Использование инструкции FUN для определения событий оценки с помощью функций LISP.

степенное замедление темпа исполнения музыкального произведения, выполняет постепенные изменения темпа. *Пример* — программный код № 10.

Базовая реализация макросов доступна в SDL и позволяет описывать повторяющиеся структуры, такие как тремоло[1] и т.д. Для этой цели используется инструкция MAC. Первый аргумент — это имя функции LISP, а остальные — аргументы для этой функции. *Пример* — программный код № 11.

Макросы также можно использовать для управления значениями параметров с помощью функции LISP, вместо того, чтобы указывать их примечание за примечанием. *Пример* — программный код № 12.

Логистическая функция (или логистическая кривая) является общей S-образной (сигмообразной) кривой, описываемой уравнением Ферхюльста[2] (по имени впервые сформулировавшего его бельгийского математика), которое изначально появилось при изучении изменений численности населения.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Где:

x_0 — значение x средней точки сигмоиды;

[1] **Прим. автора.** Тремоло — приём игры, заключающийся в быстром многократном чередовании звуков, интервалов или аккордов.

[2] **Прим. автора.** Пьер Франсуа Ферхюльст (фр. Pierre François Verhulst) — бельгийский математик, известен работами в области моделирования численности населения.

Таблица 2. Список дополнительных инструкций определяющих управление счётом и факторами времени при использовании языка Nyquist совместно с SDL.

Инструкции по счёту (SDL)		Предписания
1	(TF args: time-factor)	Общий Фактор Времени. Все продолжительности в счёте умножаются на этот коэффициент.
2	(TIME-IN-SECONDS)	Время в секундах. Никаких аргументов не требуется. Длительность измеряется в секундах, когда темп установлен на 60.
3	(DUR lisp-expr)	Задаёт значение текущей продолжительности. В качестве аргумента можно использовать любое выражение LISP.
4	(DELTA lisp-expr)	Увеличивает значение времени.
5	(PAU lisp-expr)	То же самое, что и DELTA.
6	(INIT-INSTR string function-name)	(Имя строковой функции INIT-INSTR). Запускает инструмент. Сопоставляет строку инструмента с названием партитуры с именем функции синтеза.
7	(INSTRUMENT string)	(Строка ИНСТРУМЕНТА) Устанавливает текущий инструмент на строковый режим. Ноты, следующие этой инструкции, отображаются ***строкой инструмента*** до тех пор, пока не будет ***установлен новый инструмент***.
8	(ATTR: symbol lisp-expr)	Устанавливает значение символа в lisp-expr. Первый аргумент должен начинаться с двоеточия (: symbol). Значение этого атрибута сохраняется до тех пор, пока не настанет очередь новой инструкции ATTR.
9	(PWL-POINT: symbol lisp-expr)	Ведёт себя как ATTR. Единственное различие заключается в том, что линейная интерполяция выполняется между двумя последовательными инструкциями PWL-POINT.
10	(CH &rest pitches)	(CH &оставшиеся поля) Создает аккорд, содержащий высоту тона, заданную его аргументом и текущую продолжительность.
11	(CH1 pitch duration)	Создаёт ноту, высота и продолжительность которой задаются аргументами. Время не увеличивается, так что следующее событие начинается в то же время.
12	(FUN #'function-name &rest args)	Вызывает функцию имени функции, отправляющую аргументы в списке аргументов. Возвращаемое значение должно быть событием, которое будет добавлено в музыкальную партитуру. Это событие должно быть обработано функцией Nyquist timed-seq. Следовательно, событие должно соответствовать формату (начало-время растяжения-синтез фактора-функция-вызов)
13	(MAC macro-name &rest args)	Вызывает имя макрокоманды функции, отправляя аргументы в списке аргументов. Возвращаемое значение должно быть кодом SDL, который заменяет вызов макроса.
14	number	Число. На самом деле это остаток длительности такта числа.
15	symbol	Символ. Нота, высота тона которой задаётся символом и с текущей длительностью.
16	(pitch dur)	Нота, указанная аргументами pitch и dur.

Таблица 3. Дополнительные функции библиотеки SDL.

Функции библиотеки SDL		Предписания
1	(sdl->score score-data &optional time-marks)	Создаёт музыкальную партитуру, состоящую из (функции синтеза фактора растяжения времени начала) формат списка событий. score-data представляют собой счёт SDL. Метки времени — это символ, список свойств которого содержит метки времени, на которые следует ссылаться из данных оценки.
2	(sdl->timelabels score-data &optional time-marks)	Возвращает символ, список свойств которого содержит метки времени в метках времени, добавленных к меткам времени данных оценки. Синхронность может быть обеспечена с помощью меток времени.
3	(sdl: normalize-score-duration score)	Оценка представляет собой счёт Xmusic. Эта функция устанавливает все коэффициенты растяжения событий равными 1,0. Это предназначено для того, чтобы сделать параметры синтеза независимыми от продолжительности нот.

Таблица 4. Перечень дополнительных функций, предназначенных для счёта при составлении программного кода на языке Nyquist.

Функции счёта Nyquist (свойства и события)			Процедурные Предписания
Диалект SAL		Диалект LISP	
	event-time(event)	(event-time event)	Извлекает поле времени из события.
	event-set-time(event, time)	(event-set-time event time)	Создаёт новое событие, в котором время события заменяется временем.
	event-dur(event)	(event-dur event)	Извлекает поле продолжительности (т.е. фактор растяжения) из события.
	event-set-dur(event, dur)	(event-set-dur event dur)	Создаёт новое событие, в котором длительность (или фактор растяжения) события заменяется на dur.
	event-expression(event)	(event-expression event)	Извлекает поле выражения из события.
	event-set-expression(event, dur)	(event-set-expression event dur)	Создаёт новое событие, в котором выражение события заменяется выражением.
	event-end(event)	(event-end event)	Получает время окончания события и его продолжительность.
	expr-has-attr(expression, attribute)	(expr-has-attr expression attribute)	Проверяет, имеет ли выражение события оценки данную характеристику.
	expr-get-attr(expression, attribute [, default])	(expr-get-attr expression attribute [default])	Получает значение данной характеристики из выражения события оценки. Если атрибут отсутствует, возвращает значение по умолчанию, если оно указано, и в противном случае — nil.
	expr-set-attr(expr, attribute, value)	(expr-set-attr expr attribute value)	Создаёт новое выражение, идентичное expr, за исключением того, что атрибут имеет значение.
	event-has-attr(event, attribute)	(event-has-attr event attribute)	Проверяет, имеет ли выражение данного события оценку атрибута.
	event-get-attr(event, attribute, [default])	(event-get-attr event attribute [default])	Получает значение данной характеристики из выражения события оценки. Если атрибут отсутствует, возвращает значение по умолчанию, если оно указано, а в противном случае — nil.
	event-set-attr(event, attribute, value)	(event-set-attr event attribute value)	Создаёт новое событие, идентичное событию, за исключением того, что атрибут имеет значение.

L — максимальное значение кривой;

k — темпы логистического прироста или крутизна кривой.

FUN-инструкция позволяет определять события оценки с помощью функций LISP. Включение атрибута: mm является обязательным, если генерируемое событие является примечанием. *Пример* — программный код № 13.

Здесь make-heap — создаёт неупорядоченный массив данных. «Inicio pitch» — производит запуск поля. В дополнении к вышеизложенному автор предлагает ознакомиться с дополнительными инструкциями счёта SDL представленными в табличном виде. Данные инструкции помогут звукоинженерам и программистам по звуку настраивать процедуры счёта по собственным предпочтениям, приведённые ниже данные могут служить в качестве памятки по использованию наиболее частых действий связанных с управлением счётом (таблицы: 2,3,4).

Итак, нами рассмотрена библиотека языка Nyquist, позволяющая проводить манипуляции со счётом. Представленная на обозрение научной общественности статья, служит хорошим дополнением к практикам программирования на языке Nyquist.

Рассмотренные в рукописи функции, отвечающие за счётные характеристики, упростят понимание процессов обработки аудиоданных на языке Nyquist и дадут базовые представления инженерам по звуку в области проектирования алгоритмических композиций. Автор не сомневается, что библиотека, написанная в своё время Педро Моралесом (Pedro J. Morales), станет хорошим вспомогательным инструментом в Nyquist-программировании особенно при проведении операций, направленных на составление алгоритмических композиций и синтезированных звуковых комбинаций.

В свою очередь материалы, собранные из различных источников (в т.ч. инструкций) и уточнённых в ходе исследования автором данной статьи, послужат питательной почвой для перспективных научных исследований в данной сфере.

ЛИТЕРАТУРА

1. Таран В.В. Корректировка аудиосигнала при монтаже аудиозаписей в программной среде Audacity®, используя мультифункциональные возможности языка программирования Nyquist // Современная наука актуальные проблемы теории и практики: Серия естественные и технические науки — 2021.— № 3.— С. 155–202. [ISSN2223–2966]. (DOI 10.37882/2223–2966.2021.03.32).
2. Таран В.В. Цифровая библиотека чтения и записи аудиоданных Libsndfile: техническая структура, возможности использования и перспективы развития / В.В. Таран, Р.С. Гиляревский // Современная наука актуальные проблемы теории и практики: Серия естественные и технические науки — 2021.— № 6.— С. 131–155. [ISSN2223–2966].
3. Таран В.В. Компьютерная очистка аудиоматериала штатными средствами программы Audacity® (программно-ориентированный подход) / В.В. Таран// Современная наука актуальные проблемы теории и практики: Серия естественные и технические науки — 2020.— № 9.— С. 112–128. [ISSN2223–2966]. (DOI 10.37882/2223–2966.2020.09.37).
4. Таран В.В. Язык программирования Nyquist: настоящее время и перспективы его развития в области компьютерной аудиоинженерии и аудиоинформатики / В.В. Таран// Современная наука актуальные проблемы теории и практики: Серия естественные и технические науки — 2020.— № 4.— С. 135–153. [ISSN2223–2966]. (DOI 10.37882/2223–2966.2020.04.37).
5. Таран В.В. Компьютерный аудиосинтез штатными средствами Audacity® с возможностью имитационного дизайн-моделирования на языке Nyquist/ В.В. Таран// Современная наука актуальные проблемы теории и практики: Серия естественные и технические науки — 2020.— № 1.— С. 115–129. [ISSN2223–2966].
6. Таран В.В. Проектирование дизайна аудиопродукции в программной среде Audacity® с применением языка Nyquist/ В.В. Таран// Современная наука актуальные проблемы теории и практики: Серия естественные и технические науки — 2019.— № 10.— С. 159–171. [ISSN2223–2966].
7. Таран В.В. Сравнительный анализ качества передаваемой информации форматами MP3, OGG, AIFF, WMA для оптимального выбора трансляции в Интернете/В.В. Таран // Материалы XVII международной научно-практической конференции Академическая наука — проблемы и достижения [Технические науки], North Charleston, USA.— 2018 г., Том 1, стр.78–94 [ISBN: 978–1729559000].
8. Таран В.В. Актуальные проблемы развития аудиоинженерии в России и их философско-техническое обоснование /В.В. Таран// Материалы X международной научно-практической конференции Фундаментальная наука и технологии — перспективные разработки [Технические науки], North Charleston, USA.— 2016 г., Том 1, стр. 108–123 [ISBN: 978–1729559000].
9. Таран, В.В. АУДИОМАСТЕРИНГ: Динамика современной культуры / В.В. Таран, О.В. Шлыкова // Вестник Московского государственного университета культуры и искусств.— 2016.— № 3.— С. 84–92.
10. Morales P.J. Score Description Library — Reference Manual / Перевод на английский язык: Luis Rodr´ıguez and Pedro J. Morales/ July 25, 2007.— 15 с.
11. Компьютерная программа Nyquist IDE v.3.15 / Файл директории (C:\Users\Name\nyquist) // Полная реализация — Jesse Clark, David Hovard, David Movatt, David Deangelis, Roger B. Dannenberg.— 2002–2018. [электронный источник, компьютерная программа].
12. Simoni M., Dannenberg R., Algorithmic Composition (A Guide to Composing Music with Nyquist) / Published in the United States of America by The University of Michigan Press Manufactured in the United States of America (e-book).— 2008, 2013.— p.249. [ISBN978–0–472–02905–1].

13. Touretzky, David S. Common LISP: a gentle introduction to symbolic computation /Carnegie Mellon University /// Copyright © 1990 by Symbolic Technology, Ltd.//// Published by The Benjamin/Cummings Publishing Company, Inc.p.587 [ISBN0–8053–0492–4].
14. Seibel Peter. Practical COMMON LISP /APRESS — 2005, p.528 [ISBN1–59059–239–5].
15. Dannenberg R.B. Nyquist Reference Manual Version 3.16 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 2013–2020 WEB-version, URL: http://www.cs.cmu.edu/~rbd/doc/nyquist/ [дата обращения к электронному ресурсу: 12.03.2022].
16. Dannenberg R.B. Nyquist Reference Manual Version 3.15 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 11.08. 2018 p.276.
17. Dannenberg R.B. Nyquist Reference Manual Version 3.09 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 27.12. 2014 p.297.
18. Dannenberg R.B. Nyquist Reference Manual Version 2.36 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 05.03. 2007 p.205.

Всероссийский институт научной и технической информации РАН