ИНФОРМАТИКА

INFORMATICS

Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика. 2023.
Выпуск 3 (48)
Bulletin of Syktyvkar University.
Series 1: Mathematics. Mechanics. Informatics. 2023; 3 (48)

Научная статья

УДК 004.43
https://doi.org/10.34130/1992-2752_2023_3_49

ЯЗЫК ДЛЯ ГЕНЕРАЦИИ ТАБЛИЦ В ДОКУМЕНТАХ

Евгений Анатольевич Белых,
Юрий Валентинович Гольчевский
Сыктывкарский государственный университет
имени Питирима Сорокина, hunter_x5_95@mail.ru, yurygol@mail.ru

Аннотация. Данная работа посвящена проектированию специализированного Си-подобного языка, предназначенного для генерации сложных таблиц в документах на основе данных, извлеченных из большого числа разных источников. Предлагаемый язык имеет необходимый минимум типичных для Си-подобных языков конструкций. Язык имеет свою систему типов, выстроенную на основе таблиц и адаптированную под визуальное представление данных. Имеется ряд специализированных операторов, упрощающих работу с таблицами. Помимо этого, дизайн языка позволяет работать одинаковым образом с входными данными разных форматов при наличии сценария, преобразующего их во внутренний формат языка – CSV. В работе также приведены примеры использования языка для решения типовых задач по генерации таблиц.

Ключевые слова: электронные документы, генерация документов, языки программирования, электронные таблицы

Для цитирования: Белых Е. А., Гольчевский Ю. В. Улучшение языка для генерации таблиц в документах // *Вестник Сыктывкарского университета. Сер. 1: Математика. Механика. Информатика.* 2023. Вып. 3 (48). С. 49–71. https://doi.org/10.34130/1992-2752_2023_3_49

Article

Table generating language enhancement

Evgeniy A. Belykh, Yuriy V. Golchevskiy
Pitirim Sorokin Syktyvkar State University, hunter_x5_95@mail.ru, yurygol@mail.ru

Abstract. This paper is about designing a specialized C-like language, made to generate complex tables based on large number of various data sources. Proposed language has essential set of construction that are typical for C-like languages. Language has it's own data types system that was built around tables and was adapted for visual data representation tasks. Besides a language design allows to operate uniformly with various input data types that have corresponding shell-scripts, that convert them to language's internal format – CSV. This paper also contains some language usage examples for most common cases.

Keywords: electronic documents, document generating, programming languages, electronic tables

For citation: Belykh E. A., Golchevskiy Yu. V. Table generating language enhancement. *Vestnik Syktyvkarskogo universiteta. Seriya 1: Matematika. Mekhanika. Informatika* [Bulletin of Syktyvkar University, Series 1: Mathematics. Mechanics. Informatics], 2023, no 3 (48), pp. 49–71. (In Russ.) https://doi.org/10.34130/1992-2752_2023_3_49

Введение

Современные документы, насыщенные самыми разнообразными данными, представляют достаточно сложную конструкцию. Ситуация может усугубляться тем, что информацию нужно черпать из разных источников (различные серверы, другие документы в сети, локальные файлы и т. д.). Некоторые проблемы и вопросы интеграции информации из нескольких источников данных рассмотрены в работе [1]. Далее

информацию требуется представить в наиболее понятном для пользователя виде, визуализировать ее, что часто является не совсем тривиальной задачей [2]. Это может приводить к необходимости обработки, хранения и генерации достаточно сложных данных и таблиц. Например, в работе [3] представлен подход для работы со сводными таблицами при обработке статистических данных, в [4] авторы предлагают общее представление документа табличной формы на основе XML и рассматривают систему анализа структуры на основе графов. При этом собственно конечной визуализации представления и генерации документов, содержащих сложные таблицы, интегрирующие данные из разных источников, уделяется недостаточно внимания. Практически отсутствуют удобные инструменты, позволяющие эффективно решать подобные задачи.

Ранее, в работе [5] был предложен *язык макроподстановок для генерации документов* и вспомогательный *язык для генерации сложных таблиц*, используемый совместно с основным макроязыком. Дизайн макроязыка достаточно прост и в ходе дальнейшей работы не претерпел больших изменений. Работа над развитием языка для генерации таблиц привела к необходимости внесения существенных изменений в его дизайне.

Основной идеей данного языка является «*склейка*» сложной таблицы из более простых таблиц. Для склейки необходимо последовательно записывать команды, возвращающие таблицы. Была реализована возможность использовать блоки, содержащие либо команды, либо другие блоки. Результатом вызова блока являлась генерация таблицы. Можно было указать направление склейки и дополнительную опцию для склейки с растяжением.

Были предложены команды двух типов — *подстановки* из *источников данных* и встроенные команды (`print` и `sources`). Подстановки из источников представляют собой вызов функции и имеют от одного до трёх аргументов. Первый аргумент указывал идентификатор сервера, с которого необходимо получить данные. Второй аргумент — столбец, который необходимо извлечь. И третий содержал условие фильтрации, которому должна была соответствовать строка, чтобы из нее были извлечены данные.

Помимо прочего, была доступна опция многократного вызова блоков. Эта опция позволяла указать команду, возвращающую таблицу, а также переменную, которая будет содержать одну ячейку этой табли-

цы при очередном вызове блока. Блок вызывается столько раз, сколько ячеек имеется в таблице, полученной при вызове указанной команды.

Несмотря на то что для несложных задач подобный дизайн достаточен, он имеет ряд ограничений. В частности, отсутствует возможность использования уже построенной таблицы в последующих командах, так как пользователь не может определять свои переменные, за исключением переменной для многократного вызова. По этой причине нельзя реализовать аггрегатные функции, такие как подсчёт суммы по строкам и столбцам, подсчёт количества ячеек и т. д.

Также в языке невозможны вычисления над данными, полученными из источников, — в нём отсутствуют арифметические операции, невозможно ветвление. В языке нет математических функций, так как при таком дизайне в них нет большого смысла — из-за отсутствия пользовательских переменных, их все-равно можно было бы использовать только для статически определяемых данных.

Описанные недостатки обусловили необходимость внесения изменений в дизайн языка. Для этого необходимо внести изменения в синтаксис, направленные на то, чтобы сделать его более похожим на Си-подобные языки программирования общего назначения.

Подобный переход позволяет значительно расширить возможности языка и решить все описанные проблемы, дополнительно создав большой потенциал для дальнейшего улучшения. Также важно, что данные изменения не усложняют код, а напротив, делают его более понятным и логичным и в большинстве случаев более компактным.

Таким образом целью данной работы является изменение синтаксиса языка генерации таблиц и расширение его функционала.

Обзор необходимого функционала в других языках

При разработке дизайна нового языка необходимо учитывать возможности других Си-подобных языков, в первую очередь самого языка Си. Он должен проектироваться так, чтобы пользователь, владеющий любым другим Си-подобным языком, мог быстро научиться работе с новым инструментом.

По этой причине хорошим решением представляется сохранение идеи и общей структуры языка Си, а также самых распространённых его конструкций. В первую очередь, это разделение программы на определения функций, переменных и структур, команды, состоящие из операторов и вызовов функций, и управляющие конструкции [6].

Однако также важно учитывать, что язык Си является языком низкого уровня, предназначенным в основном для задач, где требуется максимальный контроль над ресурсами системы и доступ ко всем её возможностям. Поэтому при проектировании более узкоспециализированных языков, не требующих такого функционала, многие конструкции можно упростить.

Были изучены и проанализированы особенности дизайна языков PHP [7], AWK [8] и JavaScript [9], что позволило сделать ряд очень полезных при проектировании выводов. Первый такой вывод — это то, что система типов данных, основанная на динамической типизации, полезна далеко не во всех случаях. Её чаще всего уместно использовать в узкоспециализированных языках программирования, не задействующих сложные низкоуровневые механизмы системы наподобие управления процессами.

При проектировании системы типов данных очень важно заранее продумать правила преобразования одних типов данных в другие таким образом, чтобы это происходило максимально логично и прозрачно для программиста.

Особое внимание необходимо уделить тому, как будут вести себя различные операторы при несоответствии типов операндов — как будет выполняться неявное преобразование.

Также важный аспект при проектировании языка программирования, особенно если данный язык использует динамическую типизацию, — это переменные и их область видимости. Как и в случае с преобразованием типов данных, правила выбора области видимости переменной должны быть прозрачны для программиста. Дополнительно стоит рассмотреть возможность объявления переменных с заранее определённым типом данных, так как в некоторых случаях это может упростить отладку программ.

Помимо прочего, сделан вывод, что нецелесообразно без необходимости добавлять новые сущности и отклоняться от основных принципов, принятых при проектировании языков программирования.

И последний значимый аспект, который не относится напрямую к проектированию языков программирования, но при этом является одним из самых важных, — это сложность реализации интерпретатора или компилятора. Язык, спроектированный без учета особенностей и проблем, встречающихся при написании анализаторов текста, в итоге может оказаться трудно реализуемым на практике.

Хорошим подходом к проектированию языка программирования является построение грамматики таким образом, чтобы её можно было разобрать анализатором с настолько простым дизайном, насколько это возможно — например, анализатором, основанном на рекурсивном спуске. Также важно, сколько символов потребуется знать анализатору «наперед» при чтении кода программы из потока данных — обычно языки программирования проектируются таким образом, чтобы это число не превышало одного.

Учитывая опыт и особенности изученных языков, были приняты решения по изменению языка для генерации таблиц. При проектировании системы типов данных в значительной степени был учтён опыт языков JavaScript и AWK. И несмотря на то, что задачи, решаемые ими, несколько отличаются, этот опыт оказался достаточно полезным.

Язык заранее проектировался так, чтобы программы, написанные на нём, можно было разбирать, используя обычный LL(1)-анализатор, основанный на рекурсивном спуске [10].

Типы данных и переменные

Система типов нового языка основана на динамической типизации. Значения разных типов данных преобразуются друг к другу в зависимости от используемого оператора и *старшинства*.

Старшинство определяет итоговый тип данных для значений, которые необходимо привести к общему типу: приведение всегда выполняется к старшему. Типы данных, расположенные в порядке возрастания старшинства, выглядят так: пустое значение, целое число, число с плавающей точкой, символьная строка, таблица. Как можно заметить, старшинство типов данных определено так, чтобы при преобразовании к старшему типу терялось как можно меньше информации.

Помимо старшинства, результат неявного преобразования зависит от используемого оператора или встроенной функции, которые имеют свои ограничения на допустимые типы операндов и аргументов. Подробно такие ограничения будут описаны в разделе, посвященном операторам. Тут будут приведены только правила преобразования одного типа данных в другой:

- Пустое значение, при преобразовании его в число даёт `0`, при преобразовании в строку – пустую строку, а при преобразовании в таблицу – таблицу без ячеек.

- Число преобразуется в строку путём перевода его в символьное представление.

- Строка преобразуется в число по правилам, аналогичным тем, что используются в функции scanf языка Си [11].

- Строки и числа преобразуются в таблицу путём превращения их в таблицу с одной ячейкой, содержащей эту строку или число в качестве своего значения.

- Таблица преобразуется в строку только в том случае, если она содержит единственную ячейку, значение которой и будет являться требуемой строкой. Если же таблица содержит больше одной ячейки, такое преобразование считается ошибкой.

- Целые числа преобразуются в десятичные таким образом, чтобы они были представлены с минимальными потерями.

- Десятичные числа преобразуются в целые путём отбрасывания десятичной части.

В языке имеется возможность объявлять переменные. Они задаются так же, как и в большинстве Си-подобных языков: с помощью оператора присвоения «=», который будет описан в разделе, посвященном операторам. Специальной команды для объявления переменных нет. Как и в большинстве других языков с динамической типизацией, переменная считается объявленной после того, как ей первый раз было присвоено значение.

Переменные можно использовать везде, где требуется какое-либо значение. Все переменные в языке имеют локальную область видимости: объявленную переменную можно использовать только в теле той функции или управляющей конструкции, где она была объявлена.

Обращение к ячейкам таблиц

Поскольку, таблицы являются типом данных, требуется метод обращения к отдельным их ячейкам. Для этого был добавлен специальный оператор, по сути являющийся обобщением оператора обращения к элементу массива «`[ ]`»:

```
t[rowfilter][colfilter]   ,
```

где `rowfilter` — *фильтр строк*, `colfilter` — *фильтр столбцов*. Фильтры строк и столбцов — это обобщение индексов строк и столбцов. Каждый фильтр состоит из одного или более запросов, разделённых символом «`:`». Запрос может быть номером строки или столбца, именем, *диапазоном значений* или выражением. При извлечении данных выбираются все строки, которые соответствуют хотя бы одному запросу из фильтра строк, и все столбцы, соответствующие хотя бы одному запросу фильтра столбцов.

В дальнейших примерах будет использоваться таблица `t`, изображенная на рис. 1. Первая строка на рисунке не является частью таблицы — она содержит имена столбцов и добавлена для лучшего понимания.

id	name	surname	job	hiredate	sent	recieved
1	Fyodor	Aleksandrow	manager	23.04.2018	783	583
2	Vasiliy	Dmitriev	manager	05.08.2019	473	478
3	Ivan	Petrov	finansist	27.05.2017	674	551
4	Aleksey	Nikolaev	egnineer	12.03.2012	175	97
5	Mikhail	Sergeev	egnineer	19.09.2010	94	59

Рис. 1. Пример таблицы

Если запрос является числом, то выбирается строка или столбец с номером, соответствующим этому числу. Например, с помощью следующей команды будет выбрано значение ячейки, находящейся в 6-м столбце 1-й строки таблицы `t`:

```
t[1][6];    ,
```

а команда

```
t[2;5][6];
```

даст в результате таблицу, изображенную на рис. 2а.

Если запрос является символьной строкой, то выбирается строка или столбец с именем, соответствующим этой символьной строке. Например, с помощью команды

```
t[]["name"];
```

будет выбран весь столбец с именем «`name`» таблицы `t`.

Также запрос может являться диапазоном значений. Диапазон значений — это два числа или две символьные строки, разделённые символами «`..`». Первое значение обозначает начало диапазона, а второе — его конец. Например, при использовании команды

```
t[2..5]["name"];
```

будет выбрана подтаблица из ячеек таблицы `t`, находящихся на строках с 2-й по 5-ю в столбце «`name`», которая изображена на рис. 2b.

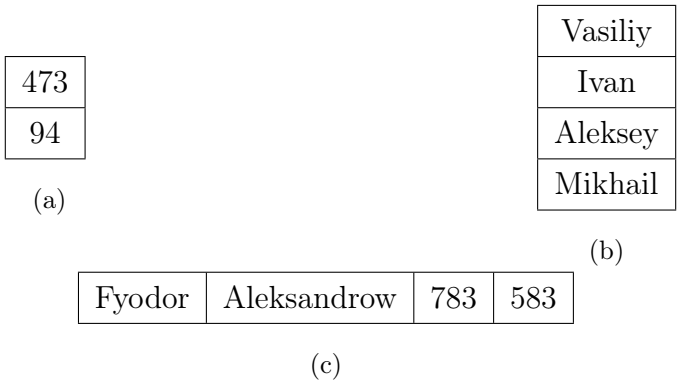

Рис. 2. Пример результата запросов

Запрос может являться выражением, которому должны соответствовать выбираемые строки. В этом случае, числа и символьные строки обозначают не номера и имена строк и столбцов, а постоянные значения. Для того чтобы идентифицировать с их помощью строки и столбцов, перед ними необходимо поместить символ «`$`».

В выражениях используются стандартные операторы сравнения, логические «и», «или» и отрицание, операции сложения, вычитания, умножения и деления, унарные «плюс» и «минус», а также вызовы встроенных функций. Логическое отрицание и все операторы сравнения, кроме равенства, обозначаются такими же символами, как и в языке Си. Логические «и» и «или», а также равенство обозначаются одиночными символами вместо двойных: т. е. это «&», «|» и «=».

Предположим, что необходимо извлечь из таблицы `t` имена, фамилии и должности работников, *фамилии* всех представителей управляющего персонала, которые отправили более 700 документов, а также тех, у которых число полученных документов составляет менее 80 % от числа отправленных:

```
t[$"job" = "manager" & ($"sent" > 700 | $"received"
  / $"sent" < 0.8)]["name".."job";"sent";"received"]   ,
```

получаем в результате исполнения кода таблицу, изображенную на рис. 2c.

Язык запросов имеет два типа встроенных функций: функции для работы с датами и функции для работы со строками. Полный список функций для работы с датами содержит функции для извлечения компонент даты (дня, месяца, года), сравнения дат, а также для нахождения разницы между датами в днях, месяцах или годах. Среди функций для работы со строками на данный момент имеются функции `match`, `cat`, `substr`, `length`.

Операторы

Данный язык, как и любой Си-подобный язык, имеет ряд операторов. Некоторые из них стандартные, такие как арифметические и логические операторы, операторы сравнения, а некоторые — специализированные, такие как склейка таблиц.

Оператор присвоения «`=`» используется для присвоения значения выражения переменной. Имеет правую ассоциативность, т. е. если выражение содержит несколько идущих подряд операторов присвоения, они будут выполняться справа налево.

Арифметических операторов в языке четыре: «`+`», «`-`», «`/`», «`*`». Они обозначают соответственно сложение, вычитание, деление и умножение. Эти операторы можно использовать только с числами — если один из операндов не является целым числом или числом с плавающей точкой, его тип преобразуется к типу второго операнда, если оба операнда не являются числами, они преобразуются в числа с плавающей точкой. В случае, когда оба операнда являются числами, преобразование типов выполняется по старшинству.

Операторы сравнения в данном языке стандартны. Всего их шесть: «`==`» (равно), «`!=`» (не равно), «`<`» (меньше), «`>`» (больше), «`<=`» (меньше или равно), «`>=`» (больше или равно).

Данные операторы не могут работать с таблицами: если один из операндов является таблицей, его тип преобразуется к типу второго операнда, если оба операнда являются таблицами, они преобразуются в строки, иначе преобразование выполняется по старшинству. Все операторы сравнения, как и в Си, возвращают целое число — «`1`», если

равенство или неравенство выполняется, и «0», если оно не выполняется.

Часто операторы сравнения используются вместе с логическими операторами: «&» (логическое «и»), «|» (логическое «или») и «!» (логическое отрицание). Логическое «и» имеет приоритет, выше которого только вызов функции и извлечение подтаблицы.

Данные операторы могут использоваться со значениями любого типа. Сначала все операнды проверяются на истинность. После этого на основе полученных значений истинности ищется результат, зависящий от конкретного логического оператора.

Истинность значения находится следующим образом:

- Пустое значение никогда не является истиной.
- Целое число является истиной, если оно содержит значение, отличное от 0.
- Число с плавающей точкой не рекомендуется проверять на истинность. В этом случае число сравнивается с нулём и считается истиной, если оно достаточно близко к нему.
- Строка считается истиной, если она содержит хотя бы один символ.
- Таблица всегда является истиной.

В языке имеется тернарный условный оператор. Данный оператор работает таким же образом, как и в большинстве других Си-подобных языков: если первый операнд — истина, возвращается второй операнд, иначе возвращается третий операнд.

Также в языке присутствует оператор конкатеции строк: он обозначается символом «~». Его приоритет находится сразу после арифметических операторов. Оба его операнда приводятся к строкам, и в результате также возвращается строка.

Помимо операторов, описанных выше, в языке имеется несколько специфичных операторов, предназначенных для «склейки» таблиц — составления таблицы из двух более простых таким образом, что вторая таблица присоединяется к первой с одной из сторон.

Всего таких оператора два — «<-» и «^». Оператор «<-» обозначает склейку по горизонтали, т. е. когда вторая таблица присоединяется

к первой справа, а оператор «^» — склейку по вертикали, когда вторая таблица присоединяется снизу.

Все операторы склейки имеют левую ассоциативность и равный между собой приоритет. Для примера выведем количество отправленных каждым сотрудником документов, разбив их по должностям, как это показано в листинге 1. В результате при подстановке шаблона `empdoc_t` будет получена таблица, изображенная на рис. 3.

Листинг 1

Пример использования склейки таблиц

```
label={lst:nextto}]
#table empdoc_t
e = (t[$"id" = 4]["name"] ~ t[$"id" = 4]["surname"]
  <- t[$"id" = 4]["sent"]) ^ (t[$"id" = 5]["name"]
  ~ t[$"id" = 5]["surname"] <- t[$"id" = 5]["sent"]);

m = (t[$"id" = 1]["name"] ~ t[$"id" = 1]["surname"]
  <- t[$"id" = 1]["sent"])
  ^ (t[$"id" = 2]["name"] ~ t[$"id" = 2]["surname"]
  <- t[$"id" = 2]["sent"]);

f = (t[$"id" = 3]["name"] ~ t[$"id" = 3]["surname"]
  <- t[$"id" = 3]["sent"]);

return e <- m <- f;
#end
```

Aleksey Nikolaev	175	Fyodor Aleksandrov	783	Ivan Petrov	674
Mikhail Sergeev	94	Valiliy Dmitriev	473		

Рис. 3. Пример результата склейки

Операторы склейки позволяют задавать дополнительные опции. Эти опции записываются как `a <- b : options` и `a ^ b : options`, где `options` — это строка, содержащая опции, а `a` и `b` — таблицы для склейки.

На данный момент единственная имеющаяся опция склейки — это `span`, указывающая, что при несовпадении размеров склеиваемых таблиц по оси, противоположной оси склейки, необходимо не добавлять новые ячейки, а растягивать существующие. Данная опция полезна при генерации заголовков таблиц и много раз будет использована в дальнейших примерах.

Функции

Вызов функции выполняется так же, как и в языке Си, и выглядит так:

```
function(args);    ,
```

где `args` — аргументы функции, разделённые символом «`,`». Можно использовать как встроенные функции, так и определять свои.

Все встроенные функции имеют заранее заданный тип данных для каждого аргумента, если в качестве аргумента передаётся значение другого типа, выполняется неявное преобразование типов.

Полный список встроенных функций на текущий момент содержит три типа функций: аггрегатные функции, функции для работы со строками и функции для работы с датами. Среди аггрегатных функций имеются следующие: `csum`, `rsum`, `cmin`, `rmin`, `cmax`, `rmax`. Они используются для выполнения операций над группами ячеек таблиц.

Функции для работы со строками и с датами похожи на те, что используются в выражениях при обращении к ячейкам таблиц. Реализованы функции для работы со строками: `match`, `substr` и `length`. Функции для работы с датами представлены следующим списком: `day`, `month`, `year`, `week`, `weekday`, `rudate`, `weekfrom`, `weekto`, `monthend`, `datecmp`, `datediff`, `dateadd`.

Для определения пользовательских функций предлагается следующий синтаксис:

```
function func([args])
{
[commands];
}    ,
```

где `args` — разделенные символом «`,`» аргументы, которые могут отсутствовать, а `commands` — выполняемые при вызове функции команды. Для возврата значения из пользовательской функции используется команда

```
return r;    ,
```

завершающая выполнение функции и заставляющая её вернуть значение `r`. Если эта команда отсутствует, после своего завершения функция возвращает значение `EMPTY`.

Управляющие конструкции

В рассматриваемом в данной работе языке были добавлены две управляющие конструкции — «`if`» и «`for`», а также две специальные команды «`continue`» и «`break`».

Конструкция `for` позволяет выполнять итерацию по строкам или столбцам таблицы:

```
for (v in table [direction]) {
[commands]
}    ,
```

где `table` — таблица, по которой выполняется итерация или выражение, дающее эту таблицу, `v` — переменная, которая будет содержать строку или столбец таблицы `table`, соответствующую выполняемому шагу итерации, `[direction]` — ключевое слово, указывающее направление итерации, а `[commands]` — команды, содержащиеся в теле конструкции `for`. В случае, когда тело конструкции `for` содержит только одну команду, символы «`{`», «`}`» можно не использовать. Имеется два направления итерации: по строкам — в этом случае вместо `[direction]` указывается слово `rows` или не указывается ничего, и по столбцам — вместо `[direction]` указывается слово `cols`. Оно же является направлением по умолчанию.

Имеются команды для безусловного перехода в циклах — `continue` и `break`. Команда `continue`, находящаяся в теле цикла, вызывает немедленное завершение текущей итерации и переход к следующей, а команда `break` вызывает немедленное завершение всего цикла. Если любая из этих команд находится вне цикла `for`, обработка шаблона завершается с ошибкой.

Условный оператор `if`, присутствующий в языке, задаётся так же, как и в других Си-подобных языках:

```
if ([condition]) {
[commands 1]
}
else {
[commands 2]
}    ,
```

где `[condition]` — условие, которое проверяется на истинность. Если `[condition]` — истина, выполняются команды `[commands 1]`, иначе выполняются команды `[commands 2]`. Как и в случае с `for`, если блок содержит только одну команду, символы «`{`» и «`}`» для него можно не использовать.

Также слово `else` вместе со вторым блоком может отсутствовать — тогда, в случае, если `[condition]` не является истиной, обработка шаблона продолжится без выполнения дополнительных команд.

Для примера в листинге 2 приведён шаблон таблицы, содержащей статистику отправленных разными сотрудниками документов. Сотрудники сгруппированы по должностям. Для каждого сотрудника указано количество отправленных им документов, а также показатель, определяющий, какой процент от общего числа отправленных документов был отправлен именно этим сотрудником. Также можно обратить внимание на использование оператора `if` для исключения возможного деления на `0`. Результат показан на рис. 4.

Листинг 2

Пример использования функций, конструкции `for` и конструкции `if`

```
label={lst:for}]
#table empdoc_t
function jobstat(t, job, s)
{
r = job ^ ("employee" <- "sent" <- "%") : "span";

for (id in t[$"job" = job]["id"]) {
e = t[$"id" = id][];

r = r ^ (e[]["name"] ~ e[]["surname"] <- e[]["sent"]
<- int(100.0 * e[]["sent"] / s + 0.5) ~ "%");
}

return r;
}

if ((s = csum(t)[]["sent"]) == 0)
return "Cannot compute: no documents were sent.";
```

```
e = jobstat("engeneer", s);
m = jobstat("manager", s);
f = jobstat("financist", s);

...
#end
```

engeneer			manager			financist		
employee	sent	%	employee	sent	%	employee	sent	%
Aleksey Nikolaev	175	8%	Fyodor Aleksandrov	783	36%	Ivan Petrov	674	31%
Mikhail Sergeev	94	5%	Valiliy Dmitriev	473	22%			
total	269	12%		1256	57%		674	31%

Рис. 4. Пример таблицы, построенной с помощью функций, конструкции `for` и конструкции `if`

Подстановки

Входные данные для данного языка предполагается передавать в одном из трех видов: в виде простого текста, в виде CSV-файла либо в виде shell-сценария, способного принимать аргументы. На данный момент в качестве основного способа передачи списка всех источников данных предлагается использовать первый аргумент командной строки интерпретатора, указывая в нем для каждого источника данных файл, формат и имя.

Например, строка, приведенная в листинге 3, определяет три источника данных: CSV-файл `emps.csv`, к которому можно обращаться по имени `employees`, простой текстовой файл `repname.txt`, доступный по имени `reportname`, и shell-сценарий `deplist.sh`, который можно вызвать по имени departmens.

Листинг 3

Пример списка источников данных для интерпретатора (разбит на несколько строк для удобства чтения)

```
label={lst:sourcelist}]
"employees:csv:emps.csv; \
reportname:text:repname.txt;\
departments:csv:deplist.sh"
```

Каждому источнику данных в языке соответствует либо переменная, либо функция — в зависимости от его типа. Имя этой переменной или функции идентично имени источника данных. Подстановка, соответственно, выполняется либо посредством обращения к переменной, либо с помощью вызова функции.

В случае с источником типа `text`, „переменная“, к которой происходит обращение, является символьной строкой, содержащей подставляемый текст, а в случае с источником типа `csv` — таблицей, куда помещено содержимое соответствующего табличного файла.

При подстановке из источника типа `script` и `sql` вызываемая „функция“принимает в качестве единственного аргумента таблицу с одной строкой. Для источника типа `script` эта строка содержит аргументы командной строки, передаваемые shell-сценарию, а для источника типа `sql` — параметры для подключения к базе данных.

В предыдущих разделах подразумевалось, что столбцы таблицы имеют имена. Как правило, эти имена получаются именно при подстановке. Для `csv`-данных имена столбцов берутся из первой строки, а для sql-запросов имена столбцов задаются с помощью оператора `as` языка sql.

Во всех предыдущих примерах использовалась таблица `t`. Предполагалось, что эта таблица уже имеется. В дальнейших примерах данные, аналогичные тем, что содержались в таблице `t`, будут подставляться из внешних источников.

Предположим, что имеется несколько баз данных с одинаковой структурой, находящихся в разных городах и содержащих информацию о сотрудниках организации, работающих в этих городах. Данные, необходимые для доступа к ним, находятся в CSV-файле `servers.txt`, содержимое которого выглядит следующим образом:

```
name;user;password;path
Syktyvkar;admin;123pas;168.192.1.1
Ukhta;sysdba;strongpas;168.192.16.1
Vorkuta;sysdba;!ps#wd;168.192.32.1
```

Доступ к этому файлу осуществляется через источник данных `servers`:

```
#input csv servers : servers.txt
```

Необходимо сформировать из содержимого этих баз данных таблицу, подобную той, что строилась в предыдущих разделах, дополни-

тельно сгруппировав сотрудников по городам, в которых они работают. Данные из баз будут извлекаться с помощью подстановки из источника данных `empdoc` типа `sql`, приведённого в листинге 4, где `datefrom` и `dateto` — передаваемые обработчику шаблона параметры, задающие дату начала и конца периода, за который генерируется отчёт.

Листинг 4

Пример объявления источника данных

```
#input sql empdoc
select
e.id as 'id', e.name as 'name', e.surname as 'surname',
j.name as 'job', e.hiredate as 'hiredate',
sum(iif(dr.id is not null, 1, 0)) as 'recieved',
sum(iif(ds.id is not null, 1, 0)) as 'sent'
from
employee j join job j on j.eid = e.id
left join document dr on dr.reciever = e.id
left join document ds on ds.sender = e.id
where
d.date between '*{datefrom}' and '*{dateto}'
group by
e.id, e.name, j.name, e.surname, j.name, e.hiredate;
#end
```

Новый шаблон таблицы приведён в листинге 5. Полученная в результате таблица изображена на рис. 5.

Листинг 5

Пример использования источника данных

```
#table empdoc_t
...
r = EMPTY;
for (srv in servers) {
t = empdoc(srv);
...
r = r ^ (srv[]["name"]
^ (e <- m <- f ^ ("total" <- csum(e)[][2]
<- int(100.0 * csum(e)[][2] / s + 0.5) ~ "%"
```

```
<- "" <- csum(m)[][2]
<- int(100.0 * csum(m)[][2] / s + 0.5) ~ "%"
<- "" <- csum(f)[][2]
<- int(100.0 * csum(f)[][2] / s + 0.5) ~ "%"))
: "span");
}
#end
```

Syktyvkar								
engeneer			manager			financist		
employee	sent	%	employee	sent	%	employee	sent	%
Aleksey Nikolaev	175	8%	Fyodor Aleksandrov	783	36%	Ivan Petrov	674	31%
Mikhail Sergeev	94	5%	Valiliy Dmitriev	473	22%			
total	269	12%		1256	57%		674	31%
Ukhta								
engeneer			manager			financist		
employee	sent	%	employee	sent	%	employee	sent	%
Dmitriy Vasilyev	15	1%				Aleksandr Ilyin	871	85%
Nikolay Alekseev	12	1%						
Sergey Dmitriev	122	12%						
total	149	15%		0	0%		871	85%
Vorkuta								
engeneer			manager			financist		
employee	sent	%	employee	sent	%	employee	sent	%
Ilya Fyodorov	24	3%	Petr Ivanov	693	97%			
total	24	3%		693	97%		0	0%

Рис. 5. Пример таблицы, построенной на основе множественных sql-запросов

Заключение

В результате анализа применимости предложенного в работе [5] языка для генерации таблиц была спроектирована его новая версия, что позволяет решать больше разнообразных задач. Код, написанный на языке для генерации таблиц, при этом стал более логичным и компактным.

Введена система типов данных, подстроенная вокруг таблиц, как основного типа, определены правила, по которым типы данных преобразуются друг к другу. Также введены правила неявного преобразования на основе старшинства типов данных.

В язык были добавлены основные операторы — сложение, вычитание, умножение, деление, унарный минус, операторы сравнения и логические операторы. Также были добавлены оператор для присвоения значения переменной и сами переменные. Появились специфические операторы, необходимые для данного языка, — это операторы для «склейки» таблиц по вертикали и горизонтали, а также оператор для конкатенации строк.

Язык предлагает управляющие конструкции, такие как циклы и команды для безусловного перехода в них, а также условный оператор. Для условного оператора был определен набор правил, учитывающих появившиеся типы данных, по которым определяется истинность управляющего выражения.

Реализована возможность обращения к отдельным ячейкам таблиц, содержащихся в переменных и возвращаемых функциями. Для этого добавлен специальный оператор, являющийся расширением оператора языка Си, использующийся для обращения к элементам массива. Его главным отличием является возможность обращения к группам ячеек, для чего применяется специальный язык выражений, описывающих условия, которым должны соответствовать строки или столбцы, чтобы быть выбранными.

Добавлен ряд встроенных функций — функции для работы с датами, функции для работы со строками, а также аггрегатные функции, позволяющие выполнять операции над группами ячеек таблиц. Также была добавлена возможность определения пользовательских функций, имеющаяся во многих других Си-подобных языках.

Подстановки данных из внешних источников теперь определены либо как глобальные переменные, содержащие результат таких подстановок, либо как функции, его возвращающие, — в зависимости от типа конкретного источника.

Дальнейшую работу планируется направить на доработку синтаксиса, которая даст языку структуры и ассоциативные массивы.

Список источников

1. **Fong J., Shiu H. and Cheung D.** A relational-XML data warehouse for data aggregation with SQL and XQuery // *Software-Practice & Experience*. 2008. 38 (11). Pp. 1183–1213. DOI: 10.1002/spe.868.

2. **Badam S. K., Liu Z. and Elmqvist N.** Elastic Documents: Coupling Text and Tables through Contextual Visualizations for Enhanced Document Reading // *IEEE Transactions on Visualization and Computer Graphics*. 2019. Vol. 25. No 1. Pp. 661–671, Jan. 2019, DOI: 10.1109/TVCG.2018.2865119.

3. **Okada M., Takaba M., Kaihara S., Okada M.** Formal Representation of Summary Tables for Health Care Statistical Database Management // *Computers and Biomedical Research*. 1998. Vol. 31. No 6. Pp. 426–450. DOI: 10.1006/cbmr.1998.1491.

4. **Amano A., Asada N.** Graph grammar based analysis system of complex table form document // *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR 2003)*. Edinburgh, Scotland, 2003. Pp. 916–920.

5. **Белых Е. А., Гольчевский Ю. В.** Подход к проектированию языка подстановок для генерации электронных документов, содержащих сложные таблицы // *Вестник Удмуртского университета. Математика. Механика. Компьютерные науки*, 2019. Т. 29. Вып. 3. С. 422–437.

6. **Brian W. Kernighan, Dennis M. Ritchie.** The C Programming Language. 2nd edition. Englewood Cliffs, New Jersey: Prentice Hall, 1988. 272 p.

7. PHP: PHP Manual. Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, Jakub Vrana And several others, Peter Cowburn (eds). 2021. URL: https://www.php.net/manual/en/index.php (дата обращения: 01.06.2023).

8. The Open Group Base Specifications Issue 6, awk [Электронный ресурс]. URL: https://pubs.opengroup.org/onlinepubs/000095399/utilities/awk.html (дата обращения: 01.06.2023).

9. ECMA-262. 12th edition. June 2021 [Электронный ресурс]. URL: https://262.ecma-international.org/12.0/ (дата обращения: 01.06.2023).

10. **Alfred V. Aho, Monica S. Lam, Ravi Sthi, Jeffrey D. Ullman**. Compilers: principles, techniques, and tools. 2nd edition. Boston: *Addison-Wesley*. 2006. 1010 p.

11. The Open Group Base Specifications Issue 6, scanf. URL: https://pubs.opengroup.org/onlinepubs/009695399/functions/fscanf.html (дата обращения: 01.06.2023).

References

1. **Fong J., Shiu H. and Cheung D.** A relational-XML data warehouse for data aggregation with SQL and XQuery. *Software-Practice & Experience*. 38 (11). Pp. 1183–1213. DOI: 10.1002/spe.868.

2. **Badam S. K., Liu Z. and Elmqvist N.** Elastic Documents: Coupling Text and Tables through Contextual Visualizations for Enhanced Document Reading. *IEEE Transactions on Visualization and Computer Graphics*. 2019. Vol. 25. No 1. Pp. 661–671, Jan. 2019, DOI: 10.1109/TVCG.2018.2865119.

3. **Okada M., Takaba M., Kaihara S., Okada M.** Formal Representation of Summary Tables for Health Care Statistical Database Management. *Computers and Biomedical Research*. 1998. Vol. 31. No 6. Pp. 426–450. DOI: 10.1006/cbmr.1998.1491.

4. **Amano A., Asada N.** Graph grammar based analysis system of complex table form document. *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR 2003)*. Edinburgh, Scotland, 2003. Pp. 916–920.

5. **Belykh E. A., Golchevskiy Yu. V.** An approach to designing a substitution language for generating electronic documents containing complex tables. *Vestnik Udmurtskogo Universiteta. Matematika. Mekhanika. Komp'yuternye Nauki* [Bulletin of Udmurt University. Mathematics. Mechanics. Computer science]. 2019. Vol. 29. Issue 3. Pp. 422–437. (In Russ.)

6. **Brian W. Kernighan, Dennis M. Ritchie.** *The C Programming Language*. 2nd edition. Englewood Cliffs, New Jersey: Prentice Hall, 1988. 272 p.

7. *Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, Jakub Vrana And several others, Peter Cowburn (eds), 2021 PHP: PHP Manual.* Available at: https://www.php.net/manual/en/index.php (accessed: 01.06.2023).

8. *The Open Group Base Specifications Issue 6, awk.* Available at: https://pubs.opengroup.org/onlinepubs/000095399/utilities/awk.html (accessed: 01.06.2023).

9. *ECMA-262 12th edition, June 2021.* Available at: https://262.ecma-international.org/12.0/ (accessed: 01.06.2023).

10. **Alfred V. Aho, Monica S. Lam, Ravi Sthi, Jeffrey D. Ullman** *Compilers: principles, techniques, and tools – 2nd edition.* Boston: *Addison-Wesley*, 2006. 1010 p.

11. *The Open Group Base Specifications Issue 6, scanf.* Available at: https://pubs.opengroup.org/onlinepubs/009695399/functions/fscanf.html (accessed: 01.06.2023).

Сведения об авторах / Information about authors
Белых Евгений Анатольевич / Evgeniy A. Belykh
аспирант / postgraduate student
Сыктывкарский государственный университет имени Питирима Сорокина / Pitirim Sorokin Syktyvkar State University
167001, Россия, г. Сыктывкар, Октябрьский пр., 55 / 167001, Russia, Syktyvkar, Oktyabrsky Ave., 55

Гольчевский Юрий Валентинович / Yuriy V. Golchevskiy
к.ф.-м.н, доцент, заведующий кафедрой прикладной информатики / Ph.D. in Physics and Mathematics, Associate Professor, Head of Applied Informatics Department
Сыктывкарский государственный университет имени Питирима Сорокина / Pitirim Sorokin Syktyvkar State University
167001, Россия, г. Сыктывкар, Октябрьский пр., 55 / 167001, Russia, Syktyvkar, Oktyabrsky Ave., 55

Статья поступила в редакцию / The article was submitted 20.06.2023
Одобрено после рецензирования / Approved after reviewing 12.09.2023
Принято к публикации / Accepted for publication 15.09.2023