Компьютерные инструменты в образовании, 2016
№ 6: 33–41
УДК: 004.42
http://ipo.spb.ru/journal

ВИРТУАЛЬНАЯ МАШИНА ДЛЯ ПРОЕКТА РУСИ

Терехов А.Н. [1], Митенев А.В.[1], Терехов М.А.[1]
[1]СПбГУ, Санкт-Петербург, Россия

Аннотация

В статье описана структура памяти и система команд виртуальной машины проекта РуСи. Объяснения, почему выбрано то или иное решение, будут полезны в лекциях и практических занятиях по курсу CS240 «Трансляция языков программирования». Описываемый материал уже дважды был применен в лекциях и практике для студентов третьего курса математико-механического факультета СПбГУ и показал свою методическую ценность. Авторы надеются, что эта статья будет полезна и студентам других вузов, начинающих свое знакомство с таким важным предметом, как трансляторы.

Ключевые слова: *язык C, транслятор, виртуальная машина, переносимость трансляторов, эффективность кода.*

Цитирование: Терехов А.Н. , Митенев А.В., Терехов М.А. Виртуальная машина для проекта РуСи // Компьютерные инструменты в образовании, 2016. № 6. С. 33–41 .

1. ВВЕДЕНИЕ

Первоначально проект РуСи [1] предназначался исключительно для обучения программированию школьников. Постепенно рамки этого проекта расширились: сначала оказалось, что на примере этого проекта удобно проводить занятия со студентами матмеха по технике трансляции, затем этим проектом заинтересовались военные и другие заказчики, которым была важна надежность создаваемых программных систем, которая в РуСи поддерживалась отказом от адресной арифметики, обязательным контролем индексов в массивах, да и просто подробной сигнализацией об ошибках пользователя. Не была забыта и одна из первых идей использования РуСи для программирования роботов. Данная статья является в каком-то смысле продолжением статьи [2] и описывает структуру виртуальной машины, используемой в проекте РуСи, но не ограничивается этим, а вводит читателя в круг основных понятий трансляции.

2. ИСТОРИЯ ВИРТУАЛЬНЫХ МАШИН

Виртуальные машины были придуманы Никлаусом Виртом в то время, когда он стажировался в Стэнфордском университете (в конце 60-х — начале 70-х гг.). Тогда ему понадобилось реализовать транслятор с им же придуманного языка Паскаль сразу на много машин, и, вместо того чтобы реализовывать кодогенератор в каждую машину самостоятельно, он придумал некую виртуальную, выдуманную машину [3], написал компилятор с Паскаля в коды этой выдуманной виртуальной машины, а потом реализовал интер-

претатор этой виртуальной машины на каждой целевой ЭВМ, Оказалось, что это намного эффективнее с точки зрения времени на разработку. Да, действительно, виртуальная машина работает несколько медленней за счет накладных расходов на дешифрацию кодов, но если хорошо продумать архитектуру виртуальной машины, то эти накладные расходы становятся незначительными, и, кроме того, появляется хорошая возможность отладки, которая не во всем возможна на реальной ЭВМ без специальной аппаратной поддержки — например, остановка по записи в какую–то ячейку и т. п. Поэтому, когда мы начали проект РуСи, сразу решили, что будем генерировать код не какой-то конкретной ЭВМ, а код выдуманной нами виртуальной машины, с тем чтобы не зависеть от целевой платформы.

В нашем коллективе накоплен многолетний опыт использования виртуальных машин. Мы массово применяли эту идею еще в конце 70-х, начале 80-х годов, о чем есть даже публикация 1990 года в академическом журнале «Автоматика и телемеханика» [4], который переводился на многие языки мира (а наша первая публикация на эту тему на конференции в Таллине была еще в 1984 году [5]), то есть наши публикации появились на много лет раньше, чем первая публикация по Java. Все основные идеи платформы Java были реализованы нами тогда: транслятор, отладчики, интерпретаторы виртуальной машины, даже аппаратная реализация в виде троированного управляющего вычислительного комплекса «Самсон». Поэтому виртуальная машина для РуСи оказалась сильно похожей на систему команд нашего комплекса Самсон, который мы разработали в 1987 году, а на вооружение в РВСН он был принят в 1992 году [6].

3. УСТРОЙСТВО ПАМЯТИ

Итак, начнем с описания структуры памяти. Память виртуальной машины представляется в виде массива целых `mem` от 0 до какой-то большой величины. В РуСи используются как `INT` и `LONG`, так и `FLOAT` и `DOUBLE`, но с помощью небольшого ухищрения (процедура `memcpy`), нарушающего контроль типов, можно помещать в эту память как целые, так и вещественные числа. Чтобы не удваивать набор команд, `INT` и `FLOAT` так же, как и `LONG` и `DOUBLE` занимают по 8 байтов, для виртуальной машины экономия памяти совершенно не важна, а пользователям, знающим стандартный язык С, все-таки привычнее иметь как `INT` и `LONG`, так `FLOAT` и `DOUBLE`.

В начале массива `mem` размещается код программы, полученный в результате трансляции из РуСи в коды виртуальной машины. На исполняемую команду указывает счетчик команд `pc` — Program Counter. Реально он указывает на следующее слово, так оказалось удобнее, то есть если мы исполняем команду по адресу a, то `pc` содержит значение $a+1$. Сразу за командами начинается область данных. Сначала идут глобальные данные, и на начало этой области глобальных данных указывает регистр g (global), термином «регистр» мы пользуемся исключительно по традиции, это обычная целая переменная. Каждая переменная получает свою ячейку, структура занимает ровно столько слов, сколько нужно для размещения ее полей. Чуть больше надо сказать о массивах. Массив представляется одной ячейкой, в которой хранится адрес нулевого элемента массива в динамическом стеке. Элементы глобальных массивов хранятся в `mem` после всех глобальных переменных.

Сразу за глобальными данными начинается стек статик функций. (Статика — это то, что отрабатывается во время трансляции, динамика — во время счета). Каждая функция должна иметь какую-то рабочую память для размещения своих параметров (которые в С

приравнены к локальным данным) и локальных данных. Размер этой памяти рассчитывается во время трансляции и поэтому получил название статика функции. На статику текущей исполняемой функции указывает регистр `l` (local).

После статики текущей функции располагается стек для вычисления промежуточных рабочих значений.

Например, для вычисления формулы $a*b+c*d$ надо где-то сохранить значение первого произведения $a*b$, затем сохранить значение второго произведения $c*d$ и только после этого складывать. Эти значения сохраняются в стеке. Стек располагается сразу после статики текущей функции, и на верхушку стека — последнюю занятую ячейку стека — указывает специальный регистр `x` .

В момент исполнения описания одномерного массива на верхушке стека лежит N — число элементов описываемого массива, на стеке занимается N ячеек, если элементом массива является структура, то ($N*$длину структуры) ячеек, а в ячейку, соответствующую идентификатору массива в статике исполняемой функции, записывается адрес нулевого элемента. Мы решили чуть-чуть расширить язык С, разрешив описывать массивы с динамически вычисляемыми границами, аналогично в РуСи глобальные переменные можно инициировать не обязательно константами.

Понятно, что функции могут вызывать друг друга, возможны и рекурсивные вызовы функций, поэтому все статики функций связаны в список. Вызываемая функция должна содержать в начале своей служебной памяти ссылку на статику вызывающей функции. На самом деле нам потребовалось три слова для этих служебных целей.

Нулевое слово любой статики — это адрес статики функции, из которой вызвана данная.

Первое слово — это адрес функции, которая будет вызвана из этой функции (сначала там хранится ноль).

Второе слово — это значение `Program Counter` в момент вызова, с тем чтобы можно было вернуться в нужное место вызывающей функции.

Сразу после этих трех служебных слов идут параметры функции и локальные данные. Поскольку у нас всего два базовых регистра — `l` и g, мы кодируем смещение таким образом, что если смещение отрицательное, то это означает смещение от g, а если смещение положительное, то от `l`.

Итак, получается следующая структура: программа в кодах виртуальной машины, затем глобальные данные, потом цепочка статик вызванных на данный момент функций, после каждой статики стек, в том числе память для массивов данной функции, затем снова статика, стек и память для массивов, и так далее.

Еще одна вещь, раздражавшая нас в С, — индексы, которые невозможно контролировать. Например, `A[i]` не является конструкцией С, а по сути это макроподстановка типа `A+i*`шаг (шаг — это размер элемента). Если `i` не принадлежит отрезку `[0 : N-1]`, то это ошибка, причем одна из самых массовых и в то же время трудно обнаруживаемых ошибок в С, поэтому мы решили, что нижняя граница, как всегда в С, будет равна 0, а верхняя граница $N-1$ может вычисляться в динамике, а N будет храниться в ближайшей слева от нулевого элемента ячейке.

Те, кто программируют на С, привыкли к нему и при переносе программы в РуСи, ничего не заметят — подумаешь, тратится одно лишнее слово. Если программу на РуСи перенести на другой транслятор С — также никто ничего не заметит, разве что пропадет контроль, который у нас есть.

Многомерный массив в языке С — это массив массивов. Хранится одномерный массив (раньше говорили вектор Айлифа), а в каждом элементе этого массива хранится адрес массива, который является строкой (точнее — подмассивом следующего измерения). Естественно, длина строки повторяется много раз, но это незначительный накладной расход.

Суммируя сказанное, приведем два рисунка:

Рис. 1

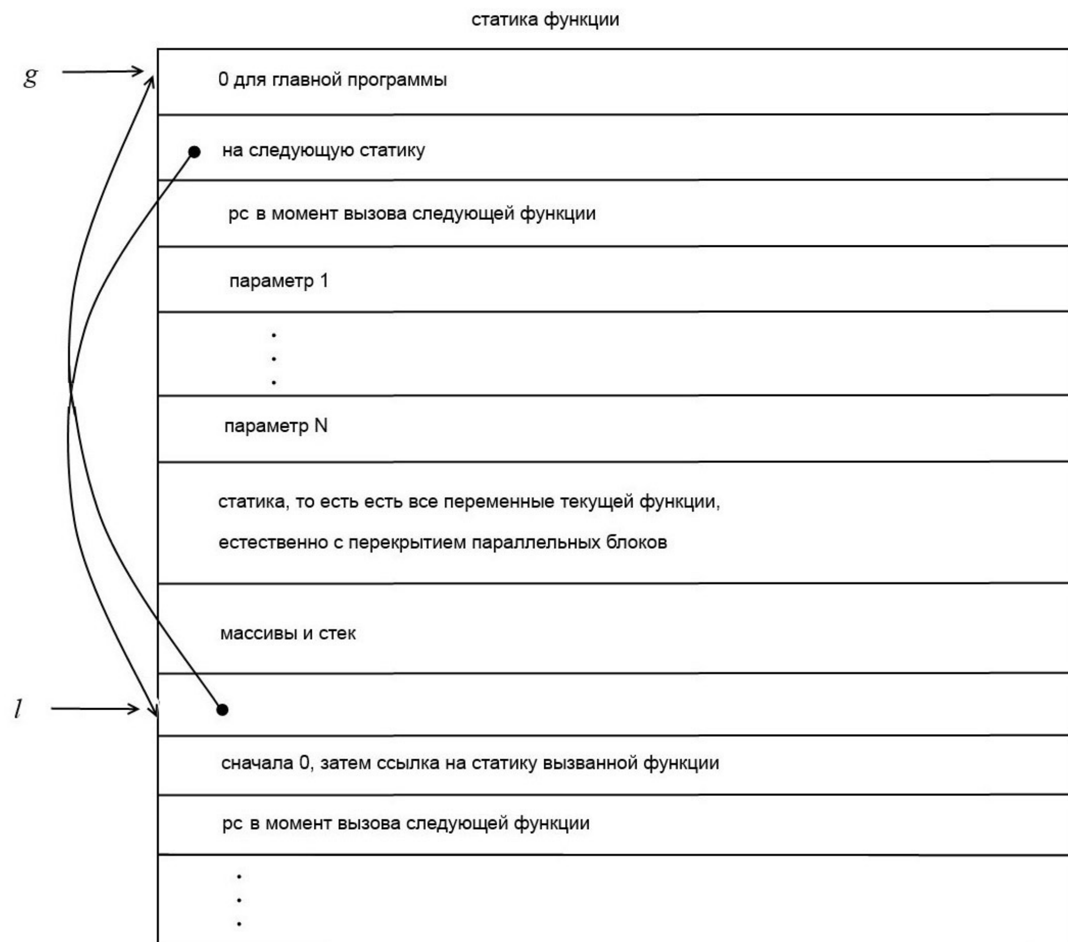

Рис. 2

В момент вызова функции новая статика начинается от следующей ячейки после x, то есть с адреса $x+1$.

Не все адреса являются статическими, то есть могут быть представлены в виде база плюс смещение. Если написано «INT a» или «FLOAT b», понятно, что адрес переменной вычисляется как база (l или g — смотря, где описаны a и b) плюс смещение от этой базы, причем смещение можно определить во время трансляции. Но если возникает вырезка элемента массива A[i] или A[i][j], то, естественно, такой адрес во время трансляции мы вычислить не можем, он вычисляется только во время счета. A[i] — выдает значение адреса i-го элемента на верхушке стека. Соответственно, есть команда SLICE, которая снимает с верхушки стека индекс, затем адрес нулевого элемента, складывает его с индексом, умноженным на шаг (параметр команды), и кладет результат на верхушку стека.

Например, a+= выражение, если a — просто статическая переменная, то это одна команда, а если a[i]+= выражение, то это другая команда. В первом случае мы знаем базу + смещение, а во втором случае динамически вычисленный адрес берется из стека.

4. КОМАНДЫ ВИРТУАЛЬНОЙ МАШИНЫ

Команд получилось довольно много — больше ста. Большую часть занимают операции языка С — в этом языке на удивление много стандартных операций, что хорошо, из-за этого зачастую программа на С получается гораздо лаконичнее, чем программа, скажем, на Паскале. Одному из авторов пришлось переводить большую программу с геологическими расчетами с Паскаля на С — заметно, насколько компактнее программа на С, в основном, за счет многих стандартных операций. Для всех операций С есть соответствующая команда виртуальной машины. Если есть команда «+=» или «-=» для целых чисел, то такая же команда есть для чисел с плавающей запятой, их кодировка отличается просто прибавлением 50 (например код команды «+» для целых чисел — 13, значит, 63 — это код такой же команды для вещественных). Обычно этого никто не видит, кроме автора интерпретатора виртуальной машины, просто это оказалось удобной особенностью. Еще одна причина для размножения команд — это операции с присваиваниями. В первом варианте РуСи было так, что после вычисления выражения a+=b на стеке оставалось значение результата (a, увеличенное на b). Если это выражение (a+=b) используется в какой-то более широкой формуле, то все проходит нормально, а если после него стоит «;», то значение приходится снимать со стека.

Мы создали специальную команду DECX (decrement x), которая сбрасывает значение с верхушки стека. Оказалось, что это была одна из самых частых команд. С одной стороны, никто не требует особенной эффективности от виртуальной машины, на то она и виртуальная, но, поскольку этих команд было много, читать было довольно противно, и мы решили сделать такой анализатор, который использовал особый вариант команд, не оставляющих значений на стеке, если они не нужны. Коды этих команд больше, чем коды остальных команд на 200 (то есть если у какой-то операции код на 200 больше, чем у стандартной, то значение операции не остается на стеке). Повторимся, пришлось раздваиваться вначале для операций с вещественными числами, потом еще раз раздваиваться для операций с присваиванием из-за разного источника адреса, а потом еще раз для операций с присваиванием, оставляющих значение на стеке или нет. В результате команда DECX исчезла совсем.

Таким образом, команд получилось довольно много, но проблем это не доставляет. Интерпретатор — вещь простая, написан один раз, переносился на роботы, на персоналки с Windows и Linux, а также на Mac. Зато программы получаются вполне симпатичными, их приятно читать глазами — это важный аргумент для трансляторщика. Еще 50 лет назад научный руководитель А.Н. Терехова Г.С. Цейтин отвечал на некоторые наши возражения, мол, не обязательно тут или тут оптимизировать, такими словами: «Вы знаете, это просто читать противно». Тогда А.Н. Терехов подсмеивался над его словами, а сейчас по прошествии многих лет считает, что Г.С. Цейтин был прав — очень важно, как программа выглядит с точки зрения человека, читающего её, а не с точки зрения тупой ЭВМ, которая молотит и исполняет программу.

Нам осталось рассказать о том, какие бывают команды, не являющиеся кодами операций языка С. Таких команд немного.

Есть команда описания массива — `DEFARR`. В момент исполнения этой команды на верхушке стека хранятся N чисел. N — это размерность массива, а каждое из чисел на стеке — это количество элементов по данному измерению. Если это число меньше или равно 0, то происходит динамическая ошибка — «неправильное число элементов», то есть число элементов должно быть больше нуля. У команды `DEFARR` есть 3 параметра: N — размерность массива, шаг, то есть длина элемента в словах (на самом деле, шаг может быть больше 1 только у массива самой младшей размерности, если тип элемента — структура) и смещение ячейки памяти, куда надо записать адрес нулевого элемента массива в целом (то есть массива самой старшей размерности).

Для массивов можно использовать инициализацию — для каждого измерения выписать нужное количество выражений через запятую в фигурных скобках. Все значения этих выражений будут загружены на стек (без каких-то разграничителей по измерениям — просто подряд), а потом команда `ARRINIT` распределит их по правильным местам. У этой команды три параметра: размерность массива, общее количество элементов инициализации, смещение ячейки описания массива.

Аналогичная команда есть и для описания структур `STRUCTINIT`, у нее два параметра: общее количество элементов инициализации и смещение первой ячейки описания структуры.

Есть команда `LI (Load Immediate)`. Ее операндом является число. Эта команда может загружать как целые, так и вещественные числа (просто двоичный код в следующем слове после команды). Плавающее число, как и целое, занимает 8 байтов — просто как двоичный код без всякого видового контроля. По коду операции мы знаем, целый операнд или вещественный, поэтому при операции с вещественными аргументами, вначале из mem значение копируется в рабочую ячейку типа `DOUBLE` (либо `rf`, либо `lf` — соответственно, правый или левый операнд), исполняется операция над этими ячейками, а потом, опять же с помощью memcpy результат копируется в mem. При печати ассемблерного кода, поскольку команда `LI` не знает, какого типа операнд, для удобства отладки, на всякий случай, печатаются и целый, и вещественный вариант.

Команда `LOAD` — у нее один операнд: смещение (еще раз повторим, либо от `l` при положительном смещении, либо от `g` — при отрицательном). Команда берет значение ячейки из памяти (локальной или глобальной) и кладет его на стек. Этой команде тоже все равно — целое значение или вещественное, на верхушку стека просто кладется двоичный код.

Команда `STOP` используется ровно один раз в конце программы, с тем чтобы закончить работу виртуальной машины при выполнении программы в кодах виртуальной машины.

FUNCBEG обозначает начало кода функции. Это немного странная команда, она реально даже никогда не исполняется, просто в С любая исполняемая программа должна содержать ровно одну функцию MAIN, и эта функция MAIN может стоять где угодно. Поэтому, когда транслируются функции, они связываются в цепной список в кодах. Когда встречается функция MAIN, она отмечается, чтобы исполнение началось именно с нее. Заодно проводится и проверка, не встречается ли функция MAIN более чем 1 раз. То есть FUNCBEG нужна как раз для того, чтобы в начале исполнения пробежаться по списку функций до функции MAIN.

Команда LA (Load Address) — операндом является смещение (как обычно, от l или g), и на стек кладется не содержимое ячейки по этому адресу, а сам адрес.

CALL1 и CALL2 — это вызов функции. Для вызова функций применяется прием, который мы использовали еще 40–50 лет назад в Алголе 68, чтобы два раза не копировать параметры. Например, когда идет вызов f (x, y+z), сначала вычисляются аргументы на стеке вызывающей функции (на верхушке стека от регистра x), а потом вызывается функция f, и ее разметка с перекрытием накладывается на стек вызывающей функции. Оказывается, что параметры уже лежат на месте.

Таким образом, CALL1 пропускает те самые 3 служебных слова, выстраивает цепочки, и дальше идет вычисление параметров на верхушке стека нормальным путем (стек уже продвинут на три ячейки), в момент исполнения команды CALL2 l указывает на вызывающую процедуру, и в ней уже есть цепной адрес статики вызываемой функции. В этот момент l передвигается на статику вызываемой функции и запоминается pc.

Функция выдает свое значение точно так же на верхушке стека. Для возврата из функции есть две команды — RETURNVAL и RETURNVOID (VOID — когда значения нет). При исполнении команды возврата все восстанавливается, как было при вызове, переходим к тому l, который был, на верхушке стека по адресу x остается значение функции, из которой выходим командой RETURNVAL, или ничего не остается, если была команда RETURNVOID.

Команда B (branch) — это безусловный переход. Операндом является индекс в массиве mem, где расположена команда виртуальной машины, куда необходимо совершить переход.

BE0 — условный переход по значению стека 0 (его еще называют переход по false), используется в while и if. Если с верхушки стека снимается 0, то переход произойдет. А если снимается любое ненулевое значение, то управление проваливается на следующую команду.

Команда BNE0 (not equal 0) действует ровно наоборот. Переход произойдет, если на верхушке стека было ненулевое значение, и провалится на следующую команду, если там был 0.

Команда SLICE с одним параметром — шаг. Она работает так: под верхушкой стека есть адрес нулевого элемента массива r, а в самой верхушке значение индекса i. Проверяется, что индекс не может быть меньше нуля и больше или равен, чем число элементов (а число элементов хранится слева от нулевого элемента), и на верхушку стека кладется r+i*шаг, то есть адрес *i*-го элемента.

WIDEN — этот термин пришел из Алгола 68 — преобразование из целого в вещественное. Значение в верхушке стека преобразуется из целого в вещественное (например когда вещественной переменной присваивается целое).

WIDEN1 — значение под верхушкой стека преобразуется из целого в вещественное.

`DOUBLE` — раздвоить верхушку стека. Тоже довольно хорошая команда, используется в реализации оператора `SWITCH`.

Есть еще коды команд операций стандартных функций (просто в ANSI C) — их не так много.

Есть несколько команд для получения значений цифровых и аналоговых датчиков (для роботов).

Список литературы

1. https://github.com/andrey-terekhov/RuC (дата обращения: 12.11.2016).
2. *Терехов А.Н.* Инструментальное средство обучения программированию и технике трансляции // Компьютерные инструменты в образовании, 2016. № 1. С. 36–47.
3. *Вирт Н.* Алгоритмы и структуры данных. М.: Мир, 1989.
4. *Матиясевич Ю.В., Терехов А.Н., Федотов Б.А.* Унификация программного обеспечения микро-ЭВМ на базе виртуальной машины // Автоматика и телемеханика, 1990. № 5. С. 168–175.
5. *Матиясевич Ю.В., Терехов А.Н.* 16-разрядная виртуальная ЭВМ, ориентированная на АЯВУ. Программирование микропроцессорной техники // Сб. «Микропроцессорная техника», Таллин, 1984. С. 69–74.
6. *Терехов А.Н.* УВК «Самсон» — базовая ЭВМ РВСН // Труды SORUCOM-2011 (Вторая международная конференция Развитие вычислительной техники и ее программного обеспечения в России и странах бывшего СССР), 2011. С. 282–286.

Поступила в редакцию 12.11.16, окончательный вариант — 13.12.16.

Computer tools in education, 2016
№ 6: 33–41
http://ipo.spb.ru/journal

VIRTUAL MACHINE FOR RuC PROJECT

Terekhov A.N.[1], Mitenev A.V.[1], Terekhov M.A.[1]
[1]SPbSU, Saint-Petersburg, Russia

Abstract

The memory structure and the command system of the virtual machine of the RuC project are described. The explanations why a particular solution is chosen would be useful in lectures and practical exercises at CS240 “Programming languages translatin” course. This material has already been used twice in lectures and practice for third-year students of the Faculty of Mathematics and Mechanics of St. Petersburg State University and has shown its methodological value. The authors hope that this article will be useful to students of other universities, starting their acquaintance with such an important subject as translators.

Keywords: *C programming language, translator, virtual machine, translator portability, code efficiency.*

Citation: Terekhov, A., Mitenev, A. & Terekhov, M., 2016. “Virtual’naya mashina dlya proekta RuSi” [“Virtual Machine for RuC project”], *Computer tools in education*, no. 6, pp. 33–41.

Received 12.11.16, the final version — 13.12.16.

Andrey N. Terekhov, professor, Head of Software Engineering Chair, SPbSU; 198504, Saint-Petersburg, Peterhof, Universitetsky pr. 28, Software Engineering Chair, a.terekhov@spbu.ru
Alexey V. Mitenev, Software Engineering Chair, SPbSU, a.mitenev@2012.spbu.ru
Mikhail A. Terekhov , Software Engineering Chair, SPbSU, st054464@student.spbu.ru

Терехов Андрей Николаевич,
доктор физико-математических наук,
профессор, заведующий кафедрой
системного программирования СПбГУ;
198504, Санкт-Петербург, Петергоф,
Университетский пр. 28, кафедра
системного программирования,
a.terekhov@spbu.ru

Митенев Алексей Владимирович,
студент кафедры системного
программирования СПбГУ,
a.mitenev@2012.spbu.ru

Терехов Михаил Андреевич,
студент кафедры системного
программирования СПбГУ
st054464@student.spbu.ru

© Наши авторы, 2016.
Our authors, 2016.