ИНСТРУМЕНТАРИЙ ПОЛЬЗОВАТЕЛЯ СИСТЕМ MATHEMATICA И MAPLE

В.З. Аладьев [a], **В.К. Бойко** [b]

[a] E-mail: **aladjev@yandex.ru;** International Academy of Noosphere; Tallinn, Estonia

[b] E-mail: **boiko@grsu.by;** Grodno State University; Grodno, Belarus

Аннотация. *Рассмотрен ряд вопросов расширения функциональной среды двух наиболее развитых на сегодня систем компьютерной математики – Mathematica и Maple, решение которых позволяет не только довольно существенно расширить сферу приложения этих систем при программировании различных задач и, прежде всего, системного характера, но и устраняет ряд существенных недостатков стандартных средств систем. Статья снабжена пакетом для системы Mathematica и библиотекой для системы Maple.*

Abstract. ***V.Z. Aladjev, V.K. Boiko. The user toolbox for Mathematica and Maple systems.***

A number of questions of expansion of the functional environment of two systems of computer mathematics which are most developed for today – Mathematica and Maple is considered whose decision allows not only to expand quite significantly a sphere of application of these systems at programming of different problems and, first of all, of system character, but also eliminates a number of essential enough defects of standard software of the systems. Article is supplied with a package AVZ_Package for Mathematica system and library UserLib6789 for Maple system.

Keywords: *Computer mathematics systems, Maple, Mathematica, Packages, Programming*

Системы компьютерной математики (***СКМ***) находят все более широкое применение в целом ряде областей как естественных, так и экономико–социальных наук: математика, физика, химия, информатика, техника, технология, образование и т.д. Системы такие как *Mathematica, Maple, REDUCE, MuPAD, Magma, Axiom, GAP, Sage* и ряд других все более востребованы для преподавания математически ориентированных дисциплин, в научных исследованиях и технологиях [1-16,18-22]. Здесь же следует отметить, что система *Sage* с *Freeware* лицензией ориентирована на исследования и преподавание в алгебре, геометрии, теории чисел, криптографии, числовых вычислениях и др. Основная цель разработчиков состоит в создании общедоступной альтернативы *Maple, Mathematica, Magma, MATLAB*.

СКМ являются основными инструментами для ученых, исследователей, преподавателей и инженеров. Исследования на основе *СКМ*–технологии, как правило, достаточно хорошо сочетают алгебраические методы наряду с продвинутыми вычислительными методами. В этом смысле *СКМ* – междисциплинарная область между математикой и информатикой, в которой исследования сосредоточиваются как на разработке алгоритмов для символьных *(алгебраических)* и численных вычислений и обработки данных, так и на создании языков программирования наряду с программной средой реализации подобного типа алгоритмов и базирующихся на них задач различного назначения. Признанными лидерами среди этих средств несомненно являются системы *Maple* и *Mathematica*. Естественно, дать сколько-нибудь полный анализ данным средствам в отведенных статьей рамках просто нереально, заинтересованный читатель отсылается к [1-9,19-22]. Здесь же акцентируется внимание лишь на одном аспекте – программной среде, поддерживаемой системами. Данный аспект имеет особую значимость не только для решения сугубо прикладных задач, но и, прежде всего, он достаточно важен в создании собственных средств, расширяющих стандартные средства систем и/или устраняющих их недостатки, либо дополняющих системы новыми средствами, имеющими достаточно массовый характер.

Несколько поясним данный аспект, которому в отечественной литературе уделяется, на наш взгляд, недостаточно внимания. К большому сожалению, у многих пользователей современных математических пакетов, не исключая системы компьютерной математики, бытует довольно распространенное мнение, что применение подобных средств не требует знания программирования, ибо все, что требуется для решения их задач, якобы уже есть в данных средствах, и задача сводится только к выбору требуемого средства. Такой подход к этим средствам носит в значительной степени дилетантский характер. Действительно, и система *Maple*, и система *Mathematica* способны решать большое число задач вообще без программирования в общепринятом смысле этого понятия. Для этого вполне достаточно описать алгоритм решения и разбить его на отдельные этапы, решаемые системой своими стандартными средствами. Тем не менее, имеется немало задач, для которых средств этих систем недостаточно; более того, для целого ряда важных задач требуется определенная модификация стандартных средств или доработка собственных, в ряде же случаев и вовсе требуется заменять стандартные, имеющие недостатки, на собственные средства. Все это и позволяет делать пользователю программная среда той либо иной *СКМ*. Итак, с полным основанием можно говорить, что и *Maple*, и *Mathematica* – это и системы компьютерной математики, и языки программирования высокого уровня. Именно в данном контексте мы и представим наши результаты по программированию в системах *Maple* и *Mathematica*.

Являясь в целом лидерами в своем классе, обе системы попеременно выходят в лидеры по той или иной группе составляющих их функций, зачастую беря в качестве примера своего основного конкурента. Между тем, даже на сегодня данные системы имеют свои коньки, по которым они превосходят конкурента. Так, если система *Maple* располагает лучшими на сегодня средствами для аналитического решения дифуравнений, система *Mathematica* в определенной мере превосходит первую в задачах интегрального исчисления. Между тем, обе системы просто нашпигованы различного рода ошибками, недоработками и другими неприятными моментами, даже, пожалуй, в несколько большем количестве, чем *Windows* и *Linux* вместе взятые, и в намного большем, чем следовало бы ожидать от современной системы компьютерной математики. Как правило, появляющиеся все новые версии обеих систем зачастую не только не устраняют большинство застарелых ошибок, но и привносят немало новых, вызывающих массу нареканий у пользователей. Именно по этой причине пользователь вынужден своими силами устранять некоторые из них или программировать собственные средства для своих задач. В частности, наряду со специфичными для каждой из систем ошибок и недостатков, обладают они и, порой, весьма существенными общими недостатками. Так, обе системы не поддерживают работу с важными операциями такими, как дифференцирование, интегрирование и подстановки при использовании выражений, а не символов в качестве переменных дифференцирования, интегрирования и левых частей *символьных* подстановок. Наш много летный опыт использования обоих систем в решении задач различного назначения наряду с исследованием внутренней *“кухни”* данных систем со всей определенностью говорят о весьма существенных сходствах многих элементов их концепции. По этой причине в обеих системах нами созданы средства, которые устраняют указанные недостатки и расширяют программную среду обоих систем инструментарием, полезным при решении как сугубо прикладных, так и системных задач. В этом контексте настоящая статья ориентирована, в первую очередь, на тех, кто при решении своих задач существенно использует программную среду данных систем, прежде всего, *Mathematica*.

Итак, учитывая общность назначения обеих систем, а также достаточно близкий уровень их развития, в качестве конкретного предмета рассмотрения отдаем предпочтение именно системе *Mathematica* как наиболее предпочтительной, на наш взгляд, для использования в достаточно серьезных научных исследованиях и приложениях. Между тем, отечественные пользователи отдают предпочтение *Maple*, что, на наш взгляд, можно объяснить большей сложностью хорошего освоения второй. Данный вопрос детальнее обсуждается в [4,5]. В связи со сказанным в качестве объекта рассматривается система *Mathematica* в контексте дополнительных средств, предназначенных для решения вышеуказанных проблем, общих

как для системы *Maple,* так и *Mathematica*. По своему основному назначению эти средства можно классифицировать следующим образом, а именно:

– *средства интерактивного режима системы Mathematica*
– *средства обработки выражений в системе Mathematica*
– *средства обработки символов и строчных выражений в Mathematica*
– *средства обработки последовательностей и списков в Mathematica*
– *средства расширения стандартных функций Mathematica, программной среды в целом*
– *определение процедур в программном обеспечении Mathematica*
– *определение пользовательских функций и чистых функций в среде Mathematica*
– *средства тестирования процедур и функций в среде Mathematica*
– *заголовки процедур и функций в программном обеспечении Mathematica*
– *формальные аргументы процедур и функций в среде Mathematica*
– *локальные переменные модулей и блоков; средства их обработки в среде Mathematica*
– *глобальные переменные модулей и блоков; средства их обработки в среде Mathematica*
– *атрибуты, опции и значения по умолчанию для аргументов пользовательских блоков, функций и модулей; дополнительные средства их обработки в среде Mathematica*
– *полезные дополнительные средства для обработки блоков, функций и модулей*
– *средства обработки внутренних файлов данных системы Mathematica*
– *средства обработки внешних файлов данных системы Mathematica*
– *средства обработки атрибутов каталогов и файлов данных в среде Mathematica*
– *специальные средства обработки файлов данных и каталогов в среде Mathematica*
– *средства работы с пакетами и контекстами, приписанными им*
– *средства организации пользовательских средств в системе Mathematica*.

Результатом решения указанных задач явилось создание *Библиотеки UserLIb6789* [13] для системы *Maple* и пакета *AVZ_Package* [14] для системы *Mathematica*. *Библиотека* создана для широкого круга пользователей, использующих систему *Maple* в их профессиональной работе. *Библиотека* содержит хорошо разработанное программное обеспечение *(более **850** средств)*, которое хорошо дополняет уже доступные инструменты *Maple* с ориентацией на самый широкий круг пользователей, существенно повышая удобство его использования и эффективность. Наш опыт показывает, использование *Библиотеки* значительно расширяет возможности системы *Maple*, упрощая программирование различных проблем в ее среде. При этом, исходные коды средств *Библиотеки* доступны пользователю как для просмотра, так и для модификации. Наконец, исходные коды средств *Библиотеки* вводят читателя и в эффективное, и в нестандартное программирование в системе *Maple*, достаточно хорошо раскрывая отдельные тонкости языка программирования системы.

Наш опыт использования системы *Mathematica* версий *7* **–** *10* позволил не только оценить ее преимущества относительно других подобных систем, прежде всего, системы *Maple*, но и выявил немало ошибок и недостатков, которые были устранены нами. Наконец, система *Mathematica* не поддерживает ряд функций, важных для процедурного программирования. Поэтому наш пакет *AVZ_Package* решает целый ряд подобных проблем. Пакет содержит более ***740*** средств, устраняющих ограничения целого ряда стандартных функций системы *Mathematica,* расширяющих ее программную среду новыми средствами. В этом контексте пакет может служить полезным инструментарием модульного программирования, которое весьма полезно в целом ряде приложений, чье программирование требует определенных нестандартных подходов. Пакет разработан для широкого круга специалистов и студентов естественно–научного профиля, использующих *Mathematica* на платформе *Windows*/*Linux*/*Mac*. Пакет содержит хорошо разработанное программное обеспечение, довольно хорошо дополняющее программную среду системы *Mathematica*. Последние обновления пакета и *Библиотеки* находятся соответственно по адресам [13,14] с *Freeware* лицензией.

В качестве основных предпосылок создания *Библиотеки* и пакета можно отметить:

(1) программирование собственных средств, в потенции носящих массовый характер;
(2) необходимость расширения стандартных средств на определенные случаи;
(3) устранение недостатков стандартных средств;
(4) создание средств, подобных средствам системы-конкурента (аналогов средств Maple для системы Mathematica, и наоборот).

Далее, на примере пакета *AVZ_Package* представим ряд типичных средств, расширяющих функциональность системы *Mathematica*, делая ее более эффективной при решении целого ряда приложений, прежде всего, носящих системный и достаточно массовый характер. В связи с обширностью представленной выше классификации внимание акцентируется нами на некоторых средствах, наиболее существенно расширяющих систему *Mathematica*.

Операции над выражениями. *Выражение* – базовое понятие *Mathematica,* для обработки которых система располагает широким набором средств, из которых важную роль играют правила подстановок в обработке структур, поддерживаемых системой *(функции, модули, строки, списки и др.)*, а также при программировании средств, расширяющих стандартные системные средства. Между тем, уже функция *ReplaceAll* имеет достаточно существенные ограничения относительно замены подвыражений очень простого вида, например:

```
In[2915]:= (c + x^2)/x^2 /. x^2 –> a
Out[2915]= (a + c)/x^2
In[2916]:= Substitution[(c + x^2)/x^2, x^2, a]
Out[2916]= (a + c)/a
```

С целью решения ряда проблем подобного типа была создана процедура *Substitution,* чей вызов *Substitution*[*x, y, z*] возвращает результат замены в выражении ***x*** всех вхождений подвыражения ***y*** на выражение ***z;*** если ***x*** – произвольное выражение, то в качестве второго и третьего аргументов, определяющих правила замен в формате *y* **–>** *z*, одинарная замена или их список, закодированных в виде {*y, z*} или {*y1, y2, …, yn*}, {*z1, z2, …, zn*} выступают соответственно; при этом, подстановки выполняются последовательно в порядке, который определяется при вызове процедуры. На основе этой процедуры и использованных в ней приемов были созданы средства, существенно расширяющие функции *D,Integrate,Replace, ReplaceAll* и др. Следующий фрагмент представляет исходный код процедуры *Substitution* с примерами сравнительных результатов применения системных и наших средств:

```
In[2968]:= Substitution[x_, y_, z_] := Module[{d, k = 2, subs, subs1},
             subs[m_, n_, p_] := Module[{a, b, c, h, t}, If[! HowAct[n], m /. n –> p, {a, b, c, h} =
                                                First[{Map[ToString, Map[InputForm, {m, n, p, 1/n}]]}];
              t = Simplify[ToExpression[StringReplace[StringReplace[a, b –> "(" <> c <> ")"],
                                                h –> "1/" <> "(" <> c <> ")"]]];
            If[t === m, m /. n –> p, t]];
             subs1[m_, n_, p_] := ToExpression[StringReplace[ToString[FullForm[m]],
                                        ToString[FullForm[n]] –> ToString[FullForm[p]]]];
            If[! ListQ[y] && ! ListQ[z],
            If[Numerator[y] == 1 && ! SameQ[Denominator[y], 1], subs1[x, y, z], subs[x, y, z]],
            If[ListQ[y] && ListQ[z] && Length[y] == Length[z],
            If[Numerator[y[[1]]] == 1 && ! SameQ[Denominator[y[[1]]], 1],
                                        d = subs1[x, y[[1]], z[[1]]], d = subs[x, y[[1]], z[[1]]]];
                                        For[k, k <= Length[y], k++,
            If[Numerator[y[[k]]]==1 && ! SameQ[Denominator[y[[k]]], 1],
                                    d = subs1[d, y[[k]], z[[k]]], d = subs[d, y[[k]], z[[k]]]]]; d,
                                                Defer[Substitution[x, y, z]]]]]

In[2969]:= Replace[1/x^2 + 1/y^3, {{x^2 –> a + b}, {y^3 –> c + d}}]
```

```
Out[2969]= {1/x^2 + 1/y^3, 1/x^2 + 1/y^3}
In[2970]:= Substitution[1/x^2 + 1/y^3, {x^2, y^3}, {a + b, c + d}]
Out[2970]= 1/(a + b) + 1/(c + d)
In[2971]:= Replace[1/x^2 + 1/y^3, {{1/x^2 –> a + b}, {1/y^3 –> c + d}}]
Out[2971]= {1/x^2 + 1/y^3, 1/x^2 + 1/y^3}
In[2972]:= Substitution[1/x^2 + 1/y^3, {1/x^2, 1/y^3}, {a + b, c + d}]
Out[2972]= a + b + c + d
In[2973]:= Replace[1/x^2*1/y^3, {{1/x^2 –> a + b}, {1/y^3 –> c + d}}]
Out[2973]= {1/(x^2*y^3), 1/(x^2*y^3)}
In[2974]:= Substitution[1/x^2*1/y^3, {x^2, y^3}, {a + b, c + d}]
Out[2974]= 1/((a + b)*(c + d))
In[2975]:= Integrate[(a/x^2 + 1/x^2)/(c/x^2 + 1/x^2), 1/x^2]
          Integrate::ilim: Invalid integration variable or limit(s) in 1/x^2. >>
Out[2975]= Integrate[(a/x^2 + 1/x^2)/(c/x^2 + 1/x^2), 1/x^2]
In[2976]:= Substitution[Integrate[Substitution[(a/x^2 + 1/x^2)/(c/x^2 + 1/x^2), 1/x^2, t],
                                                                          t], t, 1/x^2]
Out[2976]= (1 + a)/((1 + c)*x^2)
In[2977]:= D[(a/x^2 + 1/x^2)/(c/x^2 + 1/x^2), 1/x^2]
           General::ivar: 1/x^2 is not a valid variable. >>
Out[2977]= ∂1/x^2 (1/x^2 + a/x^2)/(1/x^2 + c/x^2)
In[2978]:= Simplify[Substitution[D[Substitution[(a/x^2 + 1/x^2)/(c/x^2 + 1/x^2), 1/x^2, t],
                                                                          t], t, 1/x^2]]
Out[2978]= 0
In[2979]:= D[(a/x^2 + 1/x^2)/(c/x^2 + x^2), 1/x^2, x^2]
           General::ivar: 1/x^2 is not a valid variable. >>
Out[2979]= ∂1/x^2,x^2 (1/x^2 + a/x^2)/(1/x^2 + c/x^2)
In[2980]:= Simplify[Substitution[D[Substitution[(a/x^2+1/x^2)/(c/x^2 + x^2), {1/x^2, x^2},
                                                          {t, t1}], t, t1], {t, t1}, {1/x^2, x^2}]]
Out[2980]= ((1 + a)*x^4*(c – x^4))/(c + x^4)^3
In[2981]:= Integrate[(a/x^2 + 1/x^2)/(c/x^2 + x^2), 1/x^2, x^2]
           Integrate::ilim: Invalid integration variable or limit(s) in 1/x^2. >>
Out[2981]= Integrate[(a/x^2 + 1/x^2)/(c/x^2 + x^2), 1/x^2, x^2]
In[2982]:= Simplify[Substitution[Integrate[Substitution[(a/x^2 + 1/x^2)/(c/x^2 + x^2),
                                                  {1/x^2, x^2}, {t, t1}], t, t1], {t, t1}, {1/x^2, x^2}]]
Out[2982]= ((1 + a)*(–c*(c – 2*x^4) + 2*(c^2 – x^8)*Log[(c + x^4)/x^2]))/(4*c^2*x^4)
```

Примеры со всей определенностью показывают, что операции замены, интегрирования и дифференцирования нашими средствами в отличие от системных выполняются корректно при использовании в качестве левых частей подстановок, переменных интегрирования и дифференцирования выражений, а не простых переменных, что в ряде приложений весьма существенно. Различные версии средств *Df, Df1, Df2, Diff1, Int, Int1,Integrate2, ReplaceAll1*, *Subs* и *Subs1* иллюстрируют приемы, полезные в ряде задач программирования в системе *Mathematica*. Данные средства в основном решают указанные проблемы, между тем, и они требуют своего дальнейшего развития. Вопросы реализации, использования, дальнейшего развития и особенностей средств данной группы достаточно детально рассмотрены в [7,8].

Вызов процедуры *ExprsInStrQ*[*x, y*] возвращает *True*, если строка ***x*** содержит корректные выражения, и *False* в противном случае. Тогда как через второй необязательный аргумент ***y*** **–** неопределенную переменную **–** возвращается список выражений, находящихся в ***x:***

```
In[2870]:= {ExprsInStrQ["n*(a+c)/(c+d)+(a+c)   ", t], t}
Out[2870]= {"n*(a+c)", "n*(a+c)/(c+d)", "n*(a+c)/(c+d)+(a+c)", "(a+c)", "(a+c)/(c+d)",
            "(a+c)/(c+d)+(a+c)", "(c+d)", "(c+d)+(a+c)"}
```

Показано [4-7], что далеко не все проблемы сравнения выражений с *образцами* решаются стандартными средствами. Для решения проблемы в более широком аспекте может быть весьма полезной процедура, чей вызов *EquExprPatt*[*x, p*] возвращает *True*, если выражение ***x*** соответствует заданному образцу ***p,*** и *False* в противном случае, например:

```
In[2793]:= Mapp[EquExprPatt, {a + b*c^5, 5+6*y^7, a+b*p^m, a+b*m^p}, a + b*x_^n_]
Out[2793]= {True, True, True, True}
```

Работа со строками. Строчные структуры являются одними из базовых в *Mathematica* и для их обработки система располагает достаточно широким набором средств. Между тем, в ряде случаев их оказывается недостаточно, в связи с чем нами был создан ряд полезных средств обработки строк [9,10]. В частности, процедура *SubStrSymbolParity* представляет особый интерес при обработке определений *блоков/функций/модулей* в строчном формате. Вызов *SubStrSymbolParity*[*x, y, z, d*] с четырьмя аргументами возвращает список подстрок строки ***x,*** которые ограничены одно-символьной строкой ***y, z*** ***(y ≠ z);*** при этом, поиск таких подстрок в строке ***x*** делается слева направо ***(d=0),*** и справа налево ***(d=1)***. Тогда как вызов *SubStrSymbolParity*[*x, y, z, d, t*] с ***5***-м необязательным аргументом *(целое* ***t>0****)* обеспечивает поиск в подстроке ***x,*** который ограничен позицией ***t*** и концом строки ***x*** при ***d=0,*** и началом строки ***x*** и ***t*** при ***d=1.*** В случае получения недопустимых аргументов вызов возвращается невычисленным, тогда как в случае невозможности извлечения требуемых подстрок вызов процедуры возвращает ***$****Failed*. Примеры ниже иллюстрируют применение процедуры:

```
In[3371]:= SubStrSymbolParity["12345{abcdfgh}67{rans}8{ian}9", "{", "}", 0]
Out[3371]= {"{abcdfgh}", "{rans}", "{ian}"}
In[3372]:= SubStrSymbolParity["12345{abcdfg}67{rans}8{ian}9", "{", "}", 0, 7]
Out[3372]= {"{rans}", "{ian}"}
```

Процедуры *SubStrSymbolParity, SubStrSymbolParity1* и *StrSymbParity* являются полезными инструментами, например, при обработке определений модулей и блоков, определенных в строчном формате. В целом ряде случаев возникает необходимость сведения к заданному числу количества вхождений в строку ее смежных подстрок. Эту задачу успешно решает процедура *ReduceAdjacentStr*, представленная следующим фрагментом.

```
In[55]:= ReduceAdjacentStr[x_ /; StringQ[x], y_ /; StringQ[y], n_ /; IntegerQ[n], z___] :=
                                                                    Module[{a = {}, b = {}, h, k,
                                                  c = Append[StringPosition[x <> FromCharacterCode[0], y,
                                                                    IgnoreCase –> If[{z} != {}, True, False]], {0, 0}]},
                  If[c == {}, x,
                                                                    Do[If[c[[k]][[2]] + 1 == c[[k + 1]][[1]],
                                                  b = Union[b, {c[[k]], c[[k + 1]]}], b = Union[b, {c[[k]]}];
                                                  a = Union[a, {b}]; b = {}], {k, 1, Length[c] – 1}];
                                                  a = Select[a, Length[#] >= n &];
                                                     a = Map[Quiet[Check[{#[[1]], #[[–1]]}, Nothing]] &,
                                                     Map[Flatten, Map[#[[–Length[#] + n ;; –1]] &, a]]];
                                                                    StringReplacePart[x, "", a]]]

In[56]:= ReduceAdjacentStr["abababcdcdxmnabmnabab", "ab", 3]
Out[56]= "abababcdcdxmnabmnabab"
In[57]:= ReduceAdjacentStr["abababcdcdxmnabmnabab", "ab", 1]
Out[57]= "abcdcdxmnabmnab"
In[58]:= ReduceAdjacentStr["abababcdcdxmnabmnababababab", "aB", 2]
```

```
Out[58]= "abababcdcdxmnabmnabababab"
In[59]:= ReduceAdjacentStr["abababcdcdxmnabmnabababab", "aB", 2, 6]
Out[59]= "ababcdcdxmnabmnabab"
```

Вызов процедуры *ReduceAdjacentStr*[*x, y, n*] возвращает строку, являющуюся результатом сведения к числу ***n≥0*** всех вхождений в строку ***x*** ее смежных подстрок ***y***. Если строка ***x*** не содержит подстроки ***y***, вызов процедуры возвращает исходную строку ***x***. Тогда как вызов процедуры *ReduceAdjacentStr*[*x, y, n, h*], где ***h*** – произвольное выражение, возвращает выше приведенный результат при условии, что при поиске подстрок ***y*** в строке ***x*** как прописные, так и строчные буквы рассматриваются эквивалентными.

В то время как вызов процедуры *DefToString*[*x*] возвращает определение объекта, чье имя ***x*** закодировано в строчном формате, в строчном формате. Фрагмент представляет исходный код процедуры *DefToString* с примером ее применения.

```
In[3242]:= DefToString[x_ /; StringQ[x]] := Module[{a = Definition[x], c,
                                                    b = ToString[Unique["agn"]]},
                                                Write[b, a]; Close[b]; a = ReadString[b]; DeleteFile[b];
                                            If[Set[c, Attributes[x]] == {}, a, b = StringPosition[a, "\r\n \r\n"][[1]];
                                                a = StringTake[a, {b[[2]] + 1, -1}] <> StringTake[a, b] <>
                                            "Attributes[" <> ToString[x] <> "] = " <> ToString[c] <> ";"];
                                                StringReplace[a, {"\r\n \r\n" -> "\r\n", "\r\n\r\n" -> ""}]]

In[3243]:= M[x_] := Module[{a = "12", b = "56"}, x^2]; M[x_, y_] := Module[{a = "12" b =
                                                "56"}, x + y]; SetAttributes[M, {Protected, Listable}]
In[3244]:= Definition[M]
Out[3244]= Attributes[M] = {Listable, Protected}
           M[x_] := Module[{a = "12", b = "56"}, x^2]
           M[x_, y_] := Module[{a = "12", b = "56"}, x + y]
In[3245]:= S = DefToString["M"]
Out[3245]= "M[x_] := Module[{a = \"123\", b = \"567\"}, x^2]
           M[x_, y_] := Module[{a = \"123\", b = \"567\"}, x + y]
           Attributes[M] = {Listable, Protected};"
In[3246]:= ClearAttributes[M, Protected]; ClearAll[M]
In[3247]:= ToExpression[S]
In[3248]:= Definition[M]
Out[3248]= Attributes[M] = {Listable, Protected}
           M[x_] := Module[{a = "12", b = "56"}, x^2]
           M[x_, y_] := Module[{a = "12", b = "56"}, x + y]
```

Наши средства обработки строк довольно широко применяются в различных приложениях наряду с целым рядом средств самого пакета *AVZ_Package* [14].

Работа со списками. *Списочные* структуры являются одними из базовых в *Mathematica*. В некоторых задачах программирования, использующих списочные структуры, возникает довольно настоятельная потребность замены значений списков, находящихся на заданных уровнях вложенности. Стандартные средства системы *Mathematica* такой возможности не предоставляют. В этой связи создана процедура, вызов которой *ReplaceLevelList*[*x, n, y, z*] возвращает результат замены элемента ***y***, находящегося на ***n***–уровне вложенности списка ***x***, на значение ***z***. В качестве аргументов ***y***, ***n*** и ***z*** могут выступать и списки, которые имеют одинаковую длину. При нарушения этого условия вызов процедуры возвращает $*Failed*. В случае отсутствия четвертого необязательного аргумента ***z*** вызов *ReplaceLevelList*[*x, n, y*] возвращает результат удаления элементов ***y***, которые находятся на ***n***–уровне вложенности списка ***x***. С исходным кодом процедуры, содержащим ряд достаточно полезных приемов,

можно ознакомиться в [14], в то время как следующий фрагмент представляет типичные примеры использования процедуры *ReplaceLevelList*.

```
In[2768]:= L := {a, m, n, a, {{b + d}, {c, p, t}, {{m, {{{g}}}, n, {{{{{gsv, vgs}}}}}}}}, d}
In[2769]:= ReplaceLevelList[L, {1, 3, 9}, {d, b + d, gsv}, {VG, Art, Kr}]
Out[2769]= {a, m, n, a, {{Art}, {c, p, t}, {{m, {{{g}}}, n, {{{{{Kr, vgs}}}}}}}}, VG}
In[2770]:= ReplaceLevelList[L, {1, 3, 9}, {a, p, vgs}]
Out[2770]= {m, n, {{b + d}, {c, t}, {{m, {{{g}}}, n, {{{{{gsv}}}}}}}}, d}
```

Процедура *ReplaceLevelList* допускает целый ряд достаточно интересных обобщений, что может послужить читателю в качестве весьма полезного практического упражнения. В то время как вызов процедуры *ElemsOnLevelList*[*x*] возвращает вложенный список, элементы которого – вложенные ***2***-элементные списки, чьими первыми элементами являются *уровни* списка ***x,*** тогда как вторыми – списки элементов ***x*** этих уровней вложенности. При этом, в случае пустого списка ***x,*** т.е. {}, возвращается {}. Фрагмент ниже представляет типичный пример использования процедуры *ElemsOnLevelList:*

```
In[2824]:= L := {a, m, n, a, {{b + d}, {c, p, t}, {{m, {{{g}}}, n, {{{{{gsv, vgs}}}}}}}}, d}
In[2825]:= ElemsOnLevelList[L]
Out[2825]= {{1, {a, m, n, a, d}}, {3, {b + d, c, p, t}}, {4, {m, n}}, {7, {g}}, {9, {gsv, vgs}}}
```

Многие вычисления удобно определять применением функций одновременно ко многим элементам списка. Система *Mathematica* обеспечивает ряд эффективных функциональных конструкций для этих целей *(функции MapAt, MapIndexed, MapThread, Thread и др.)*. Тогда как для случая вложенных списков данных средств в ряде случаев недостаточно. В связи с этим создана процедура, чей вызов *MapNestList*[*x,f,y*] возвращает результат применения ко всем уровням вложенности списка ***x*** функции ***f*** с передачей ей аргументов, определенных необязательным аргументом ***y,*** как весьма наглядно иллюстрирует следующий фрагмент:

```
In[2447]:= L := {6, 3, 8, {11, 8, 26, {{5, 7, 9}}, {{{3, 0, {5, 3, 7, {50, 90}, 2}, 2}}}}, {19, 26}}
In[2448]:= MapNestList[L, Sort, #1 > #2 &]
Out[2448]= {8, 6, 3, {26, 11, 8, {{9, 7, 5}}, {{{3, 0, {7, 5, 3, {90, 50}, 2}, 2}}}}, {26, 19}}
```

Наши средства для работы со списками достаточно широко используются как различными приложениями, так и целым рядом средств самого пакета *AVZ_Package* [14]. В целом ряде случаев программирования они представляют достаточно полезный инструментарий.

Функциональное программирование. ***Math–***язык, выражающий смешанную парадигму *функционального* и *процедурного* программирования довольно существенно поддерживает функциональное программирование. В основе его лежит понятие "*чистой функции*", для работы с которыми *Mathematica* имеет целый ряд средств. В расширение их были созданы дополнительные средства расширения функциональной парадигмы системы. Так, созданы средства для конвертации традиционной функции в чистую, и наоборот [7,8,14]. Тогда как при вполне приемлемых условиях процедура *ModToPureFunc* выполняет конвертирование модуля/блока в чистую функцию. Успешный вызов *ModToPureFunc*[*x*] возвращает имя в форме *ToString*[*Unique*[*x*]] искомой чистой функции, иначе вызов процедуры возвращает вложенный список, чей первый элемент $*Failed* определяет недопустимость конвертации, второй элемент – список типов переменных, которые явились причиной этого, и третий – список переменных этих типов. Ниже приведен пример с успешным вызовом процедуры:

```
In[3984]:= B[x_, y_] := Block[{a = 500, b = 90}, (a + b + c)*(x + y)];
           SetAttributes[B, {Protected, Listable}];  ModToPureFunc[B]
Out[3985]= "$$$B"
In[3986]:= Definition["$$$B"]
Out[3986]= Attributes[$$$B] = {Listable, Protected}
            $$$B := Function[{x, y}, (590 + c)*(x + y)
```

Эти и другие наши средства обеспечивают определенное расширение функциональной парадигмы, широко используемой системой *Mathematica* и ее приложениями [6-8,14].

Процедурное программирование **–** одна из основных парадигм *Mathematica*, в достаточно существенной степени отличающаяся от подобной парадигмы известных традиционных процедурных языков. Данное обстоятельство **–** краеугольный камень многих системных проблем, касающихся вопросов процедурного программирования в *Mathematica*. Прежде всего, подобные проблемы возникают в области различий реализации выше упомянутых парадигм в *Mathematica* и в среде традиционных *процедурных* языков. Тогда как в отличие от многих традиционных и встроенных языков встроенный *Math***–**язык располагает весьма скудными средствами для работы с процедурными объектами. В этом контексте довольно развитый набор таких средств представлен в наших книгах [6-8] и в пакете *AVZ_Package* [14]. В [6-8] достаточно детально рассматривается само понятие *"процедура"*, достаточно существенно отличающееся от традиционной парадигмы и играющее определяющую роль в процедурном программировании. При этом, указанные средства имеют отношение лишь к пользовательским процедурам и функциям, ибо определения всех системных функций *(в отличие, скажем, от Maple)* скрыты от пользователя. Приведем наиболее существенные средства, расширяющие возможности обработки процедурных объектов в *Mathematica*.

Прежде всего, вызов стандартной функции *Definition*[*x*] в целом ряде случаев возвращает определение объекта ***x*** с соответствующим ему контекстом, что при достаточно больших определениях становится плохо обозримым и менее удобным для программной обработки Для устранения данного недостатка определен целый ряд средств, позволяющих получать определения процедур/функций в *оптимизированном* формате *(без содержащихся в них контекстов)*. Здесь можно отметить такие средства как *Definition2, PureDefinition* и др. В первую очередь данные средства удобно использовать для т.н. *одноименных* объектов, т.е. объектов, имеющих одинаковые имена и разные заголовки [4-8,14], например:

```
In[2801]:= J[x_] := x; J[x_, y_] := Module[{}, x*y]; J[x_, y_, z_] := Block[{}, x*y*z]
In[2802]:= PureDefinition[J]
Out[2802]= {"J[x_] := x", "J[x_, y_] := Module[{}, x*y]", "J[x_, y_, z_] := Block[{}, x*y*z]"}
```

Таким образом, в отличие от привычных систем программирования одноименные объекты в *Mathematica* идентифицируются не их *именами*, а *заголовками*, требуя соответствующих средств для своей обработки. И такие средства были созданы [4-8,14]. Средства *Definition*-группы относятся именно к этим средствам, широко используясь как в программировании различных приложений, так и средствами самого пакета *AVZ_Package* [14].

Mathematica, располагая процедурами двух типов *(Module* и *Block)* и функциями, включая чистые, в то же время не имеет средств для идентификации объектов указанных типов. По этой причине был создан ряд средств [4-8], позволяющих определять объекты указанных типов, среди которых можно отметить такие как *BlockFuncModQ, FuncBlockModQ, Head2, FunctionQ, ModuleQ, ProcBMQ, ProcFuncTypeQ, ProcQ, PureFuncQ, QFunction* и др. Так, процедура *ProcQ1* применима и к простым, и к одноименным объектам. Вызов *ProcQ1*[*x*] возвращает *True*, если ***x*** определяет процедурный объект типа {*Block*, *DynamicModule или Module*} с единичным определением наряду с объектом, состоящим из их комбинаций, но с различными заголовками *(одноименные объекты)*. При этом, на единичном объекте или одноименном объекте ***x*** *True* возвращается лишь, если все его компоненты **–** процедурные объекты, т.е. имеют тип {*Block*, *DynamicModule и Module*}. Между тем, вызов процедуры *ProcQ1*[*x, y*] со ***2***–м дополнительным аргументом ***y*** **–** *неопределенной переменной* **–** через ***y*** возвращает простой либо вложенный список следующего формата {{*a1, a2, a3, …, ap*}, {*b1, b2, b3, …, bp*}}, где *aj*∈{*True, False*}, тогда как *bj*∈{*Block*, *DynamicModule, Function, Module*}; при этом, между элементами вышеупомянутых подсписков существует биекция, тогда как пары {*aj, bj*} *(j=1..p)* соответствуют подобъектам объекта ***x*** согласно их порядка в результате вызова *Definition*[*x*], как иллюстрирует достаточно простой фрагмент:

```
In[2831]:= B[x_, y_] := Module[{a = 500, b = 90}, (a + b + c)*(x + y)];
           B[x_] := Block[{}, x^2];  B[x_, y_, z_] := x*y*z;  {ProcQ1[B, g], g}
Out[2831]= {False, {{True, True, False}, {"Module", "Block", "Function"}}}
In[2832]:= J[x_, y_] := Module[{a = 500, b = 90}, (a + b + c)*(x + y)];
           J[x_] := Block[{}, x^2]; J[x_, y_, z_] := Module[{}, x*y*z]; {ProcQ1[J, g1], g1}
Out[2832]= {True, {{True, True, True}, {"Module", "Block", "Module"}}}
```

Средства тестирования объектов процедурно-функционального типа достаточно широко используются в программировании приложений и средствами пакета *AVZ_Package* [14].

В предположении общего определения процедуры, в частности, модульного типа

M[*x_*/; $Test_x$, *y_*/; $Test_y$, ...] **:=** Module[{*Локальные переменные*}, *Тело процедуры*]

и того факта, что конкретное определение процедуры идентифицируется не ее именем, а ее *заголовком*, был создан ряд достаточно полезных средств, обеспечивающих различные манипуляции заголовками процедур и функций и играющих важную роль в процедурном программировании и, прежде всего, программировании задач системного характера. Итак, *заголовок* процедуры/функции имеет вид "*Имя*[*Список формальных аргументов, которым приписаны средствами тестирования на их допустимость*]". И первый вопрос сводится к необходимости тестирования объектов на их допустимость в качестве заголовка, который решается процедурами *HeadingQ* ÷ *HeadingQ3,* покрывающими все основные случаи. При этом, из опыта их применения следует, что вполне достаточно ограничиться процедурами *HeadingQ*, *HeadingQ1*, покрывающими очень широкий диапазон ошибочного кодирования заголовков и возвращающими на допустимых заголовках *True*, в противном случае *False:*

```
In[3285]:= Map[HeadingQ, {"F[x_/; StringQ[x]]", "F[x/; StringQ[x]]", "F[x_; y_; z_]",
           "F[x; StringQ[x]]", "F[x_/_ StringQ[x]]", "F[x_//; StringQ[x]]", "F[]"}]
Out[3285]= {True, False, True, False, True, False, True}
```

Тогда как вызов процедуры *HeadPF*[*x*] возвращает в строчном формате заголовок объекта с именем ***x*** типа {*block, function, module*}, который активизирован в текущей сессии. При этом, для одноименного объекта ***x*** вызов возвращает список заголовков, порядок которых отвечает порядку определений, возвращаемых вызовом *Definition*[*x*], например:

```
In[3758]:= M[x_ /; x == "art"] := Module[{a = 26}, a*x];  M[x_, y_, z_] := x*y*z;
           M[x_ /; IntegerQ[x], y_String] := Module[{}, x];  M[x_String] := x;
           M[x_, y_] := Module[{a = 90, b = 500}, x + y];  HeadPF[M]
Out[3758]= {"M[x_ /; x == \"art\"]", "M[x_, y_, z_]", "M[x_ /; IntegerQ[x], y_String]",
             "M[x_String]", "M[x_, y_]"}
```

Процедура *HeadingsPF* является достаточно полезным расширением процедуры *HeadPF*.

В свете возможности существования *одноименных* процедур вопрос удаления из текущего сеанса процедуры с конкретным заголовком представляет вполне определенный интерес; эта проблема решается процедурой, чей успешный вызов *RemProcOnHead*[*x*] возвращает *"Done"*, удалив из текущей сессии процедуру/функцию *(их список)* с заголовком *(списком заголовков)* ***x,*** заданных в строчном формате, как иллюстрирует следующий фрагмент:

```
In[4253]:= M[x_ /; x == "art"] := Module[{a = 26}, a*x];  M[x_, y_, z_] := x*y*z;
           M[x_ /; IntegerQ[x], y_String] := Module[{}, x];
           RemProcOnHead[{"M[x_,y_,z_]", "M[x_/;x\[Equal]\"art\"]"}]
Out[4254]= "Done"
In[4255]:= Definition[M]
Out[4255]= M[x_ /; IntegerQ[x], y_String] := Module[{}, x]
```

Тогда как вызов *ExtrProcFunc*[*h*] возвращает уникальное имя блока/функции/процедуры, у которой в списке определений есть заголовок ***h;*** процедура характерна тем, что оставляет

все определения символа *HeadName*[*h*] без изменения, а возвращенный объект сохраняет все опции и атрибуты, приписанные символу *HeadName*[*h*]. Данные две процедуры наряду с другими средствами, поддерживающими обработку объектов на уровне их заголовков, обеспечивают эффективную обработку одноименных объектов [6-8,14].

Вопрос обработки формальных аргументов блока/функции/модуля связан, прежде всего, с вычислением списка формальных аргументов объектов указанного типа, активированных в текущем сеансе. В расширение среды *Mathematica* с этой целью созданы процедуры *Args* и *Args1*. Так, вызов *Args*[*x*] возвращает список формальных аргументов Compile/функции/блока/модуля ***x***. При этом, формат возвращаемого результата определен типом объекта ***x:***

– *список формальных аргументов с типами, приписанными им, если* ***x*** **–** *Compile функция;*
– *список формальных аргументов с тестами на допустимость фактических аргументов, соответствующих им;*
– *список мест {#1,…,#n} в строчном формате на чистой функции* ***x*** *короткого формата, тогда как на стандартной чистой функции* **–** *список аргументов в строчном формате.*

Тогда как вызов *Args*[*x, h*] со вторым дополнительным аргументом ***h*** **–** любое выражение **–** возвращает результат, подобный вызову *Args*[*x*] с тем отличием, что аргументы задаются в строчном формате, но без приписанных им типов и тестов на допустимость, например:

```
In[2784]:= V := Compile[{{x, _Real}, {y, _Real}}, (x^3 + y)^2];  H[x_] := Block[{a}, x];
 Kr := (#1^2 + #2^4 – 90*#3) &; Art := Function[{x, y}, x*Sin[y]]; P[y_] := Module[{b}, y]
In[2785]:= Map[Args, {V, Art, Kr, P, H}]
Out[2785]= {{x_Real, y_Real}, {"x", "y"}, {"#1", "#2", "#3"}, {y_}, {x_}}
In[2786]:= Mapp[Args, {V, Art, P, H}, gs]
Out[2786]= {{"x", "y"}, {"x", "y"}, {"y"}, {"x"}}
```

К *Args, Args1* примыкают процедуры *Arity, ArityBFM, ArgsBFM, ArgsTypes, TestArgsTypes,* последняя из которых позволяет за один проход выявить все недопустимые фактические аргументы тестируемого блока/функции/модуля, что не в состоянии обеспечить средства систем *Mathematica* и *Maple*. Для решения подобных задач представляют интерес также и процедуры *TestArgsTypes1, TestArgsTypes2, TestArgsCall, TestFactArgs* и *TrueCallQ* [9,14].

Аналогично ситуации с формальными аргументами блока/модуля система *Mathematica* не располагает средствами обработки локальных переменных объектов этих типов, довольно существенно сужая возможности процедурного программирования. В данной связи создан ряд средств манипуляции с локальными переменными пользовательских блоков/модулей, из которых отметим процедуру *Locals*, чей вызов *Locals*[*x*] возвращает список, элементы которого в строчном формате определяют локальные переменные блока/модуля ***x*** вместе с их начальными значениями. Вызов же *Locals*[*x, y*] со ***2***-м необязательным аргументом ***y*** **–** неопределенной переменной **–** обеспечивает, кроме того, возвращение через ***y*** простого ***2***-элементного списка или списка *ListList*-типа с ***2***-элементными подсписками, чьи первые элементы определяют имена локальных переменных блока/модуля ***x*** в строчном формате, тогда как второй элемент **–** начальные значения в строчном формате, приписанные им; в то время как отсутствие начальных значений определено символом *"No"*, например:

```
In[2830]:= M[x_, y_] := Module[{a = 90, b = 500}, (x + y)*(a + b)]; M[x_] := Block[{}, x^2];
           M[x_Integer, y_ /; ListQ[y]] := Block[{a, b = 90}, x^2];  Locals[M]
Out[2831]= {{"a", "b = 90"}, {"a = 90", "b = 500"}, "{}"}
In[2832]:= Locals[M, t]; t
Out[2832]= {{{"a", "No"}, {"b", "90"}}, {{"a", "90"}, {"b", "500"}}, "{}"}
```

В целом ряде случаев требуется определять только список имен локальных переменных независимо от их начальных значений. Эту задачу решает процедура, чей вызов *Locals1*[*x*] возвращает список имен в строчном формате локальных переменных блока/модуля ***x;*** при этом, на типичной функции ***x*** вызов обоих процедур возвращает *"Function"*.

В ряде случаев возникает необходимость динамического расширения списка локальных переменных блока/модуля, активированного в текущей сессии, без изменения самого кода объекта во внешней памяти. Данную задачу решает процедура, чей вызов *ExpLocals*[*x, y*] возвращает список локальных переменных *(возможно, с их начальными значениями)*, на которые расширены локальные переменные объекта ***x***. При этом, вообще говоря, данный список может быть меньше, чем список ***y***, заданный при вызове процедуры *(либо пустой)*, т.к. локальные переменные объекта ***x*** и его аргументы исключаются из него, например:

```
In[3832]:= Agn[x_] := Module[{a = 90, b, c}, a + x^2];
In[3833]:= ExpLocals[Agn, {"x", "a = c + d", "b", "Art = 26", "Sv", "Kr = 18"}]
Out[3833]= {"Art = 26", "Sv", "Kr = 18"}
In[3834]:= Definition[Agn]
Out[3834]= Avz[x_] := Module[{a = 90, b, c, Art = 26, Sv, Kr = 18}, a + x^2]
```

В то время как процедура *ReplaceLocals* обеспечивает динамическую замену локальных переменных блока/модуля на период текущего сеанса работы с *Mathematica*, например:

```
In[2793]:= Vsv[x_, y_] := Module[{a, b, c = 90, d, h}, Procedure body]
In[2794]:= ReplaceLocals[Vsv, {"a=73", "h=68", "b=2015"}]; Definition[Vsv]
Out[2794]= Vsv[x_, y_] := Module[{a = 73, b = 2015, c = 90, d, h = 68}, Procedure body]
```

Активация в текущем сеансе блока/модуля в область имен переменных системы добавляет все его локальные переменные. По этой и другим причинам эффективное использование локальных переменных – довольно важная задача. Между тем, в ходе программирования блоков/модулей вполне реально проявление т.н. "*избыточных*" переменных. Процедура *RedundantLocals* в определенной степени решает эту задачу, возвращая список в строчном формате "*избыточных*" локальных переменных блока/модуля, например:

```
In[2840]:= Vsv[x_, y_] := Module[{a, b, c = 90, d, h, g, t}, b = 500; d[z_] := z^2 + z + 500;
             h[t_] := Module[{}, t]; d[c + b] + x + y + h[x*y]]; RedundantLocals[Vsv]
Out[2840]= {"a", "g", "t"}
```

Вариант *RedundantLocalsM* обобщает *RedundantLocals* на случай одноименных объектов. Наконец, вызов процедуры *OptimalLocals*[*x*] возвращает имя блока/модуля ***x***, обеспечивая оптимизацию в текущем сеансе его локальных переменных в плане удаления *избыточных* и кратных локальных переменных, например:

```
In[2893]:= Vsv[x_, y_] := Module[{a, b, c = 90, d, c, v, h, a, t, p, b, q}, Procedure body]
In[2894]:= OptimalLocals[Vsv]; Definition[Vsv]
Out[2894]= Vsv[x_, y_] := Module[{c = 90}, Procedure body]
```

Подобно случаю локальных переменных несомненный интерес представляет определение существования в блоке/модуле глобальных переменных; в первом случае вопрос решается выше рассмотренными средствами, во втором – посредством процедур *Globals* и *Globals1*. Так, вызов процедуры *Globals*[*x*] возвращает список глобальных переменных в строчном формате блока/модуля ***x***, например:

```
In[2808]:= Sv[x_, y_] := Module[{a, b = 90, c = 500}, a = (x^2 + y^2)/(b + c); {z, h} = {a, b};
             t = z + h; g := h*t + 26; g]; Globals[Sv]
Out[2808]= {"g", "h", "t", "z"}
```

При этом, под *глобальными* понимаются те переменные, которые отличны от формальных аргументов и локальных переменных, но которым в теле объекта делаются присвоения по операторам {":=", "="}. В данном направлении создан целый ряд других полезных средств [8]; например, вызов процедуры *VarsInBlockMod*[*x*] возвращает ***4***–элементный список, чей первый элемент определяет список формальных аргументов, второй – список локальных переменных, третий – список глобальных переменных и четвертый – список переменных, отличных от системных переменных и имен, например:

```
In[2925]:= Sv[x_, y_Integer] := Module[{a, b = 90, c = 500}, a = (x^2 + y^2)/(b + c);
             {z, h} = {a, b}; t = z + h; g := h*t + 26*u*s; g];  VarsInBlockMod[Sv]
Out[2925]= {{"x", "y"}, {"a", "b", "c"}, {"g", "h", "t", "z"}, {"u", "s"}}
```

Процедура *BlockFuncModVars* обеспечивает структурный анализ блока/функции/модуля в контексте используемых им: *(1) системных функций*, *(2) средств пользователя*, *активных в текущем сеансе*, *(3) формальных аргументов*, *(4) локальных переменных*, *(5) активных глобальных переменных*, и *(6) пассивных локальных переменных*. В то время как процедура *GlobalToLocal* обеспечивает конвертацию процедуры в процедуру, в которой глобальные переменные исходной процедуры включены в список локальных переменных.

Mathematica поддерживает работу с опциями и значениями по умолчанию для аргументов блоков, функций и модулей пользователя, располагая соответствующим набором средств. Между тем, системные средства для этих целей имеют ряд ограничений и недостатков. В данной связи был создан ряд средств, таких как процедуры *Defaults, DefaultsQ, DefaultsM, DefaultValues1* и др., существенно расширяющих систему в этом направлении [6-8,14].

Наряду с простыми блоками/модулями *Mathematica* поддерживает работу с вложенными объектами этих двух типов любого уровня вложенности, ограниченного лишь размерами памяти. Между тем, стандартных средств для специальной обработки подобных объектов у системы нет, что стимулировало их создание [6-9,14]. В частности, процедуры *SubProc* ÷ *SubProc3* и *SubProcs* обеспечивают полезные манипуляции с подпроцедурами. Более того, структура главной процедуры предполагается корректной, если имя каждой вложенной ее процедуры указано в списке локальных переменных процедуры, ее содержащей. В такой организации главной процедуры доступ ко всем ее подпроцедурам недоступен, например, посредством функции *Definition*. Для проверки корректности была создана процедура, чей вызов *NotSubsProcs*[*x, y*] возвращает список имен в строчном формате всех подпроцедур у процедуры ***x*** *Module*-типа, которые не удовлетворяют выше указанному соглашению и чьи определения доступны в текущем сеансе после вызова процедуры ***x***. Тогда как через ***2***–й дополнительный аргумент ***y*** – неопределенная переменная – возвращается список имен в строчном формате всех подпроцедур процедуры ***x*** без учета их вложенности, например:

```
In[4806]:= B[x_, y_ /; IntegerQ[y]]:= Module[{}, Z[t_]:= Module[{}, H[p_]:= Module[{Q},
             Q[k_] := Module[{}, k]; p]; t^2*H[t] + Q[t]]; x*y + Z[x*y]]; {NotSubsProcs[B, t], t}
Out[4806]= {{"Z", "H"}, {"Z", "H", "Q"}}
```

Возврат вызовом *NotSubsProcs*[*x*] пустого списка говорит о корректности процедуры ***x***. С другой стороны, процедура *StrNestedMod* обеспечивает структурный анализ вложенных процедур *Module*-типа. Вызов процедуры *StrNestedMod*[*x*] возвращает вложенный список имен в строчном формате подпроцедур процедуры ***x;*** при этом, каждое имя расположено на соответствующем уровне вложенности. Тогда как, вызов *StrNestedMod*[*x, y*] со вторым дополнительным аргументом ***y*** – неопределенная переменная – через ***y*** возвращает список имен всех подпроцедур процедуры ***x,*** в котором первым элементом является ***x,*** например:

```
In[4815]:= Avz[x_, y_, z_]:= Module[{Art, Gal, b=6}, Art[a_]:= Module[{Es}, a^2 + Es[a]];
             Gal[b_, c_] := Module[{Kr, Sv}, Kr[d_] := Module[{Vg}, Vg[p_] := Module[{a, Gr},
             Gr[m_]:= Module[{}, m]; p^1]; Vg[d]*d^3]; Sv[p_]:= Module[{}, p^4]; Kr[b]*Sv[c]];
             Art[x]*Gal[y, z]];  {StrNestedMod[Avz, s], s}
Out[4815]= {{"Avz", {"Art", "Gal", {"Kr", "Sv", {"Vg", {"Gr"}}}}}, {"Avz", "Art", "Gal",
              "Kr", "Vg", "Gr", "Sv"}}
```

При отсутствии в блоке/модуле метки *Label*[*x*] для соответствующего обращения *Goto*[*x*] в момент вычисления его определения не распознается; данная ситуация распознается лишь в момент реального обращением *Goto*[*x*]. Примеры, иллюстрирующие сказанное, можно найти в [6,7]. В этой связи определенный интерес представляет процедура *GotoLabel,* чей вызов *GotoLabel*[*x*] проверяет процедуру ***x*** на формальную корректность использования ею функций *Goto* и меток *Label,* соответствующих им [8,9,14].

Отмеченные и целый ряд других средств манипуляций с блоками/функциями/модулями пользователя включены в пакет [14], образуя достаточно полезный инструментарий при работе с объектами указанного типа в программной среде системы *Mathematica*.

Средства ввода/вывода системы Mathematica. Система, ориентированная, прежде всего, на символьную обработку, имеет ряд ограничений по средствам ввода/вывода, устранение которых позволяет существенно расширить программирование задач доступа к данным. В этом направлении был создан целый ряд средств [6,8,14]. Средства системы обеспечивают доступ пользователя к файлам нескольких типов, которые могут быть условно разделены на две многочисленных группы: *внутренние* и *внешние* файлы. В рутинной работе система имеет дело с ***3*** различными типами внутренних файлов, среди которых отметим файлы с расширениями {***"nb"***, ***"m"***, ***"mx"***}, чья структура распознается стандартными средствами. Из средств ***1***-й группы можно отметить процедуру, чей вызов *Nobj*[*x, y*] возвращает список имен объектов, ранее сохраненных в файле ***x*** *m*-формата функцией *Save*, тогда как через второй аргумент ***y*** *– неопределенную переменную* **–** возвращается список заголовков этих объектов, например:

```
In[3819]:= Agn[y_] := 67;  Avz[x_] := 90*x + 500;  Save["C:/Temp/Obj.m", {Agn, Avz}]
In[3820]:= ClearAll[ArtKr];  {Nobj["C:\\Temp\\Obj.m", ArtKr], ArtKr}
Out[3820]= {{"Agn", "Avz"}, {"Agn[y_]", "Avz[x_]"}}
```

Для активизации в текущем сеансе объектов, находящихся в *m*-файле, ранее созданном по цепочке функций *GUI "File → Save As → Mathematica Package (*.m)"*, может применяться процедура *Aobj1*. Успешный вызов *Aobj1*[*x, y*] возвращает *Null* с выводом сообщений о тех средствах, которые были загружены из *m*-файла ***x*** и которые отсутствуют в файле. Кроме того, в качестве аргумента ***y*** могут выступать отдельный символ или их список [6-9,14]. В случае одноименного объекта ***y*** вызов *Save*[*x,y*] сохраняет в файле ***x*** все его составляющие. Для устранения подобной ситуации предложено обобщение *Save* функции в виде функции *Save1,* позволяющей сохранять одноименные объекты с конкретными заголовками. Более того, ряд созданных средств расширяет возможности обработки внутренних файлов [6,14].

Для работы с внешними файлами создан инструментарий, как расширяющий стандартные средства *Mathematica*, так и дополняющий их новыми достаточно важными средствами. В частности, вызов процедуры *FileExistsQ1*[*x*] возвращает *True*, если аргумент ***x*** определяет файл, реально существующий в файловой системе компьютера, и *False* иначе; тогда как вызов *FileExistsQ1*[*x, y*] через ***y*** *– неопределенную переменную* **–** возвращает список всех путей к найденному файлу данных ***x,*** если основной результат вызова **–** *True*, например:

```
In[2808]:= {FileExistsQ1["Cinema_2015.txt", y], y}
Out[2808]= {True, {"C:\\Temp\\cinema_2015.txt", "E:\\Temp\\cinema_2015.txt"}}
```

Тогда как процедура *SearchFile* обеспечивает поиск заданного файла данных в файловой системе компьютера.

Mathematica не располагает средствами для работы с атрибутами файлов и каталогов, что, по нашему мнению, является существенным недостатком, в первую очередь, при создании на ее основе различных систем обработки данных. Между прочим, подобных средств нет и в системе *Maple*, что стимулировало создание для нее ряда процедур {*Atr, F_atr, F_atr1, F_atr2*} [1-3], решающих эту проблему. Представленные ниже средства решают подобную проблему для системы *Mathematica*. Основной является процедура *Attrib,* обеспечивающая обработку атрибутов файлов/каталогов. Вызов процедуры *Attrib*[*f, Attr*] возвращает список *атрибутов* указанного файла/каталога ***f*** в контексте *Archive ("A"), Read-only ("R"), Hidden ("H")* и *System ("S")*. Более того, допустимы и другие атрибуты, приписанные системным файлам и каталогам; в частности, на главных каталогах устройств внешней памяти *"Drive f"*, на несуществующем каталоге/файле возвращается сообщение *"f isn't a directory or file"*. Вызов процедуры *Attrib*[*f,* {}] возвращает *Null*, отменяя все атрибуты для файла/каталога ***f***,

тогда как вызов *Attrib*[*f, {"x", "y", …, "z"}*], где x, y, z∈{"–A", "–H", "–S", "–R", "+A", "+H", "+S", "+R"}, также возвращает *Null*, устанавливая/отменяя атрибуты файла/справочника ***f:***

```
In[2823]:= Attrib["c:\\tp\\cinema", "Attr"]
Out[2823]= {"A", "S", "H", "R"}
```

```
In[2824]:= Attrib["c:\\tp\\cinema", {"-S", "-H", "-R"}]; Attrib["c:\\tp\\cinema", "Attr"]
Out[2824]= {"A"}
```

Процедуры *Attrib1* и *Attribs* **–** полезные модификации процедуры *Attrib,* из которых вторая является довольно быстродействующей. Целый ряд других средств не только существенно расширяют системные средства обработки файлов, но и позволяют выполнять обработку нестандартного характера *(восстановление файлов,удаленных средствами системы и Dos, очистка каталога Recycle Bin, обработка файла в разных потоках и др.)* [6-8,14].

Работа с пакетами пользователя. Дополнительно к встроенным средствам *Mathematica* располагает стандартным набором пакетов, содержащим определения различных объектов *(функции, глобальные переменные и др.)*; число и набор пакетов зависят от версии системы и для версии *10.2* их число равно *2619*. Пакет имеет стандартную организацию и содержит определения различных объектов, выполняющих требуемые функции. Между тем, наряду с системными пользователь имеет возможность создавать собственные пакеты, которые ориентированы на решение как сугубо прикладных, так и системных задач. Организация пакетов пользователя наряду с взаимосвязью пакетов и контекстов детально рассмотрены в [6]. Именно во многом в связи с пакетами введено понятие *контекста* для обеспечения операций с символами, представляющими различные объекты *(функции, пакеты, модули и т.д.)*, в частности, чтобы избежать возможных конфликтов с одноименными символами. Наряду с системными нами создан ряд средств для обработки контекстов [6-8]. Так, вызов процедуры *ContextToSymbol1*[*x, y, z*] возвращает *Null*, обеспечивая назначение контекста ***y*** символу ***x; 3-***й дополнительный аргумент ***z –*** строка, определяющая справку по ***x;*** при его отсутствии для неопределенного символа ***x*** справка **–** пустая строка, т.е.**""**, тогда как для определенного символа ***x*** справка имеет вид *"Help on* ***x****"*. В частности, *ContextToSymbol1* **–** достаточно полезное средство для расширения пакета пользователя, содержащегося в *mx*–файле, новыми средствами. Тогда как вызов процедуры *AllContexts*[] возвращает список контекстов, содержащихся в пакетах текущей версии *Mathematica;* вызов же *AllContexts*[*x*] возвращает *True*, если ***x –*** контекст указанного типа, и *False* в противном случае:

```
In[3844]:= AllContexts[]
Out[3844]= {"ANOVA`", "ANOVA`ANOVA`", "App`", ..., "XML`XML`"}
```

```
In[3845]:= AllContexts["URLUtilitiesLoader`"]
Out[3845]= True
```

Главными формами сохранения определений объектов являются *документ (Notebook)* и *пакет (Package)*, сохраняемые соответственно в файлах данных форматов {*cdf, nb*} и {*mx, m*}. Именно поэтому был создан достаточно развитый инструментарий для обработки этих файлов. Так, процедуры *ContextMfile* и *ContextNBfile* обеспечивают получение контекста, приписанного файлам форматов *"m"* и {*"cdf", "nb"*} соответственно, например:

```
In[3985]:= Map19[{ContextMfile, ContextNBfile}, {"c:/avz_pack.m", "c:/avz_pack.nb"}]
Out[3985]= {"AladjevProcedures`", "AladjevProcedures`"}
```

Процедура *ContextMfile* обеспечивает поиск первого контекста в *m*–файле с пакетом, тогда как с файлом может быть ассоциировано несколько контекстов. В этом случае процедура *ContextMfile1* решает вопрос в случае множественных контекстов, например:

```
In[3990]:= ContextMfile1["DocumentationSearch.m"]
Out[3990]= {"DocumentationSearch`", "ResourceLocator`"}
```

Известно [6-8], для каждого экспортируемого объекта пакета необходимо определить его применение *(usage)*, тогда как локальные объекты, расположенные в секции, в частности,

Private, будут недоступны в текущем сеансе. Для тестирования пакета, или загруженного в текущий сеанс, или расположенного в *m*–файле относительно существования локальных и глобальных объектов служит процедура *DefInPackage*, чей вызов *DefInPackage*[*x*], где ***x*** определяет *m*–файл либо контекст, приписанный пакету, возвращает вложенный список, чей первый элемент определяет контекст пакета, второй элемент – список локальных, в то время как третий элемент – список глобальных переменных пакета ***x,*** например:

```
In[4816]:= DefInPackage["C:\\AVZ_Package\\AVZ_Package.m"]
Out[4816]= {"AladjevProcedures`", {}, {"AcNb", "ActBFM", …, "$UserContexts"}}
```

В ряде случаев требуется полное удаление из текущего сеанса пакета. Частично проблема решается функциями *Clear* и *Remove*, но они не очищают списки, определенные вызовом *Contexts*[] и переменными $*Packages,* $*ContextPath*. Полностью данная проблема решается процедурой *RemovePackage*, вызов которой *RemovePackage*[*x*] возвращает *Null*, полностью удаляя из текущего сеанса пакет, определенный контекстом ***x,*** например:

```
In[3852]:= RemovePackage["AladjevProc`"]
In[3853]:= MemberQ[Flatten[{$Packages, $ContextPath, Contexts[]}], "AladjevProc`"]
Out[3853]= False
```

Поскольку *одноименные* объекты имеют различные заголовки, то в определенных случаях возникает проблема их более точной идентификации. Процедура *DiffContexts* предлагает один из таких подходов, ассоциируя компоненты, составляющие одноименные объекты с разными контекстами. Алгоритм процедуры *DiffContexts* основан на создании для каждой компоненты *одноименного* объекта пакета в *m*-файле с *уникальным* контекстом, например:

```
In[4415]:= T[x_] := x; T[x_, y_] := x*y; T[x_, y_, z_] := x*y*z
In[4416]:= DiffContexts[T]
Out[4416]= {{"T3`", "T[x_, y_, z_]"}, {"T2`", "T[x_, y_]"}, {"T1`", "T[x_]"}}
In[4417]:= Definition["T3`T"]
Out[4417]= T[T3`T`x_, T3`T`y_, T3`T`z_] := T3`T`x*T3`T`y*T3`T`z
In[4418]:= $Packages
Out[4418]= {"T3`", "T2`", "T1`", "AladjevProcedures`", "HTTPClient`", …}
```

В процессе работы со средствами загруженного пакета *(из m-файла)* возможны ситуации, когда некоторые из его активированных средствах по той или иной причине удалены из текущего сеанса или искажены. Для их восстановления может использоваться процедура, чей успешный вызов *ReloadPackage*[*x*] возвращает *Null*, т.е. ничего, обеспечив в текущем сеансе активизацию всех средств пакета, расположенного в *m*–файле ***x,*** тогда как вызов, *ReloadPackage*[*x, y*] со вторым необязательным аргументом ***y*** – списком имен – выполняет перезагрузку только средств пакета с указанными именами, а вызов *ReloadPackage*[*x, y, t*], с ***3***–м необязательным аргументом ***t*** – произвольным выражением, также возвращает *Null*, обеспечив перезагрузку всех средств пакета ***x,*** исключая средства с именами ***y,*** например:

```
In[3862]:= Clear[StrStr, Map6]; ReloadPackage1["C:\\AVZ_Package.m", {StrStr, Map6}]
In[3863]:= Definition[StrStr]
Out[3863]= StrStr[x_] := If[StringQ[x], "\"" <> x <> "\"", ToString[x]]
```

В [6] представлена простая и удобная для модификаций организация пакета пользователя, сохраняемого в *m*–файле по цепочке команд *GUI "File –> Save As –> Mathematica Package (*.m)"*. Именно таким способом организован и модифицировался пакет *"AVZ_Package.m"*. Процедура *ConvertMtoMx* обеспечивает конвертацию *m*-файла, созданного таким методом, в *mx*–файл, оптимально загружаемый в текущий сеанс. Вызов процедуры *ConvertMtoMx*[*x, y*] возвращает *Null*, обеспечивая конвертацию *m*–файла ***x,*** созданного указанным методом, в файл с тем же главным именем, но с расширением *"mx";* через ***2***–й аргумент ***y*** – *символ* – возвращается список имен в строчном формате объектов, которые содержатся в *m*–файле

x с пакетом. Тогда как вызов *ConvertMtoMx*[*x, y, z*] с ***3***–м необязательным аргументом ***z*** – *произвольное выражение* – дополнительно выгружает из текущего сеанса пакет файла ***x:***

```
In[5]:= Clear[StrStr, ProcQ]; ConvertMtoMx["C:\\AVZ_Package\\AVZ_Package.m", w]
In[6]:= Definition[StrStr]
Out[6]= StrStr[x_] := If[StringQ[x], "\"" <> x <> "\"", ToString[x]]
In[7]:= w
Out[7]= {"AcNb", "ActBFM", "ActBFMuserQ", "ActCsProcFunc", …, "$UserContexts"}
In[8]:= Length[%]
Out[8]= 742
```

Весьма важным является вопрос редактирования пакета пользователя, расположенного в *m*–файле. Его решение обеспечивает процедура *RedMfile*, вызов которой *RedMfile*[*x, n, y*] возвращает полный путь к *m*–файлу с базовым именем *FileBaseName*[*x*]<>"**$**" – результату применения к исходному *m*–файлу ***x*** операции ***y*** {*"add", "delete", "replace"*} относительно его объекта, определенного именем ***n***. Аналогичную функцию по редактированию пакета пользователя, расположенного в *mx*–файле, выполняет процедура *RedMxFile*. Интересные примеры использования данных процедур могут быть найдены в [6-9].

Организация программного обеспечения пользователя в системе Mathematica. Система *Mathematica* не имеет достаточно удобных средств организации библиотек пользователя как в случае *Maple*, создавая определенные затруднения в организации пользовательского программного обеспечения. Для сохранения определений объектов система *Mathematica* использует файлы различной организации. С вопросами сохранения объектов *(функции, модули, справки и т.д.)* можно ознакомиться в [4-9], тогда как здесь мы отметим простые средства организации пользовательских библиотек в системе *Mathematica*. Среди данных средств можно отметить процедуру *UserLib*, поддерживающую много полезных функций, подобных традиционным библиотекам. Вызов процедуры *UserLib*[*x, f*] обеспечивает ряд важных функций поддержки простой библиотеки, расположенной в файле ***x*** *txt*–формата. Вторым аргументом ***f*** выступает ***2***-элементный список, для которого допустимыми парами элементов могут быть {*"names", "list"*}, {*"print", "all"*}, {*"print", "Name"*}, {*"add", "Name"*}, {*"load", "all"*}, {*"load", "N"*}, смысл которых детально рассмотрен, например, в [8]. Опыт использования данной процедуры подтвердил ее эффективность в организации библиотек пользователя наряду с рядом других средств простой организации средств пользователя, в частности, библиотек пользователя с табличной организацией, подобной *Maple* [4-8].

Заключение. Представленные выше средства наряду с целым рядом других составляют пакет *AVZ_Package*, ориентированный, прежде всего, на тех, кто преследует задачу более глубокого понимания программирования в *Mathematica* и особенно тех пользователей, кто хотел бы перейти от пользователя к программисту либо тех, кто уже имеет определенный опыт программировании в *Mathematica*, но хочет повысить свою квалификацию. Опытные программисты *Mathematica*, возможно, также найдут тут полезную для себя информацию. Архив *"AVZ_Package.Zip"* с пакетом может быть бесплатно загружен с сайта [14]. Архив содержит пять файлов*: Archive.pdf, AVZ_Package.cdf, AVZ_Package.mx, AVZ_Package.nb* и *AVZ_Package.m*. В частности, для перлюстрации содержимого пакета можно использовать файл *Archive.pdf,* или *AVZ_Package.cdf* с *CDF Player,* либо файл *AVZ_Package.m* с любым текстовым процессором, например, *Notepad*. Подобный подход позволяет удовлетворить пользователя на различных платформах *(Linux ARM, Mac OS X, Linux, Windows)*. Сам пакет содержит более ***740*** средств, устраняющих ограничения целого ряда стандартных функций системы и расширяющих ее новыми средствами. В этом контексте пакет может служить в качестве полезного инструментария, особенно полезного в многочисленных приложениях, когда программирование задач предполагает использование нестандартных приемов. При этом, объем памяти, требуемой для пакета в *Mathematica 10.3.0 (на Windows 7 Professional, ver. 6.1.7601)*, оценивается как:

```
In[1]:= MemoryInUse[]
Out[1]= 29 658 544
In[2]:= Get["C:\\AVZ_Package\\AVZ_Package.mx"]
In[3]:= MemoryInUse[]
Out[3]= 42 103 304
In[4]:= N[(% – %%%)/1024^2]
Out[4]= 11.8682
```

т.е. в *Mathematica* пакет *AVZ_Package* требует около *11.9 M* памяти, тогда как количество средств, чьи определения находятся в пакете, доступно на основе простых вычислений:

```
In[5]:= Length[CNames["AladjevProcedures`"]]
Out[5]= 742
```

Процедура *ContCodeUsageM* обеспечивает возврат списка имен объектов, находящихся в *m*–файле с пакетом, структурно аналогичного рассматриваемому пакету, исходный код и справку по заданному объекту из *m*–файла с пакетом. Вызов функции *ContCodeUsageM*[*x*] возвращает список имен в строчном формате объектов пакета, находящегося в *m*–файле ***x,*** вызов *ContCodeUsageM*[*x,y*] возвращает справку по объекту ***y*** из *m*–файла ***x,*** в то время как вызов *ContCodeUsageM*[*x,y,z*], где ***z*** *– произвольное выражение* **-** возвращает исходный код объекта ***y*** из *m*–файла ***x*** с пакетом пользователя указанного формата, например:

```
In[1942]:= ContCodeUsageM[x_ /; FileExistsQ[x] && FileExtension[x] == "m", y___] :=
      Module[{a = StringJoin[Map[FromCharacterCode, BinaryReadList[x]]], b = {y}},
        If[Length[b] > 1 && Head[b[[1]]] == Symbol,
                    StringTake[StringReplace[StringCases[a,
        "Begin[\"`" <> ToString[b[[1]]] <> "`\"]" ~~ Shortest[W__] ~~ "End[]"][[1]],
                      "*)\r\n(*" –> "\r"], {13 + StringLength[ToString[b[[1]]]], –7}],
        If[b != {} && Head[b[[1]]] == Symbol,
                      StringReplace[StringTake[StringCases[a,
          ToString[b[[1]]] <> "::usage=" ~~ Shortest[W__] ~~ "(**)"][[1]], {1, –9}],
          "::usage=" –> "::", 1], Sort[DeleteDuplicates[Map[StringTake[#, {9, -4}] &,
              StringCases[a, "Begin[\"`" ~~ Shortest[W__] ~~ "`\"]"]]]], $Failed]]

In[1943]:= ContCodeUsageM["C:\\AVZ_Package\\AVZ_Package_2.m"]
Out[1943]= {"AcNb", "ActBFM", "ActBFMuserQ", "ActCsProcFunc", … , "$UserContexts"}
In[1944]:= ContCodeUsageM["C:\\AVZ_Package\\AVZ_Package_2.m", StrStr]
Out[1944]= StrStr::"The call StrStr[x] returns an expression x in string format if x is different
           from string; otherwise, the double string obtained from an expression x is returned."
In[1945]:= ContCodeUsageM ["C:\\avz_package\\avz_package_2.m", StrStr, 90]
Out[1945]= "StrStr[x_]:=If[StringQ[x],\"\\\"\"<>x<>\"\\\"\",ToString[x]]"
```

Процедура обеспечивает перлюстрацию пакета пользователя, имеющего вышеуказанную организацию и находящегося в *m*–файле, без его реальной загрузки в текущий сеанс. При этом, процедура использует только стандартные функции системы *Mathematica*, не требуя наличия в текущем сеансе загруженного выше упомянутого пакета *AVZ_Package* [14]. В то время как используемые процедурой приемы могут оказаться достаточно полезными при обработке строчных конструкций различного назначения. В частности, они очень полезны в задачах поиска в строках подстрок специального вида. Подобные приемы используются средствами рассмотренного пакета, имеющими дело с обработкой исходных кодов блоков и модулей в строчном формате, достаточно эффективно, порой, существенно упрощая сам процесс программирования.

Следующая довольно полезная процедура обеспечивает загрузку и выполнение в текущем сеансе пакета, находящегося в *m*–файле. Вызов процедуры *MfileLoad*[*x*] вычисляет пакет из *m*–файла ***x*,** возвращая *Null*, т.е. ничего, и делая доступными в текущем сеансе средства, чьи определения содержатся в файле ***x.*** В то время как вызов процедуры *MfileLoad*[*x, y*] со вторым необязательным аргументом ***y*** **–** *неопределенной переменной* **–** в дополнение через переменную ***y*** возвращает ***3*-**элементный список, элементы которого определяют контекст, приписанный пакету ***x,*** список главных имен объектов, определения которых находятся в пакете ***x*** и список имен с данным контекстом соответственно. В частности третий элемент может определять объекты системных пакетов, встроенных в систему как это имеет место, например, для пакета *Combinatorica*. Вызов процедуры на *m*–файле, не содержащем пакет, возвращает **$***Failed*. Фрагмент представляет исходный код процедуры с примерами:

```
In[1]:= MfileLoad[x_ /; FileExistsQ[x] && FileExtension[x] == "m",  y___] :=
                                                    Module[{a = ReadString[x], b, c, d, h, p},
                                                  b = Flatten[StringPosition[a, "BeginPackage[\""]];
                                    If[b == {}, $Failed,
    c[z_] := Select[z, ! MemberQ[{"Attributes[" <> # <> "] = {Temporary}", "Null", False},
                                                    Quiet[Check[ToString[Definition[#]], False]] &];
                                          d = StringCases[a, "BeginPackage[\"" ~~ Shortest[___] ~~ "`\"]"];
                                                  d = Map[Quiet[Check[StringTake[#, {15, –3}], Null]] &, d];
                                        d = Select[d, Complement[Characters[#], Join[CharacterRange["A", "Z"],
                                                    CharacterRange["a", "z"], {"`"}]] == {} &][[1]];
Quiet[ToExpression[StringReplace[StringTake[a, {b[[1]], –1}], {"(*" –> "", "*)" –> ""}]]];
                                                  b = Map[StringReplace[#, d –> ""] &, Names[d <> "*"]];
                                                Map[Quiet[ClearAttributes[#, {Protected, ReadProtected}]] &, b];
                                          If[{y} != {} && SameQ[ToString[Definition[y]], "Null"],
                                                    y = {d, Set[p, c[b]], Select[Complement[b, p],
    "Attributes[" <> # <> "] = {Temporary}" != Quiet[ToString[Definition[#]]] &]};, Null]]]

In[2]:= {MfileLoad["C:\\AVZ_Package\\AVZ_Package.m", g47], g47}
Out[2]= {Null, {"AladjevProcedures`", {"AcNb", "ActBFM", "ActBFMuserQ", …,
          "ActCsProcFunc", "$UserContexts"}, {"a", "b", "c", "d", "h", "k", "p", "S", "x", "y", "z"}}
In[3]:= Map[Length, {%[[2]][[2]], %[[2]][[3]]}]
Out[3]= {742, 12}
```

Следует иметь в виду, что ряд средств и *Библиотеки* для *Maple,* и пакета для *Mathematica* полностью или частично функционально дублируют друг друга, иллюстрируя различные приемы программирования, полезные при разработке многих приложений как системного, так и прикладного характера. При этом, отдельные средства могут быть использованы при программировании собственных проектов в качестве самостоятельных составляющих или вспомогательных средств, упрощающих программирование. Опыт использования пакета для *Mathematica* и *Библиотеки* для *Maple* со всей уверенностью доказал как их полезность при освоения продвинутых приемов программирования в системах *Maple* и *Mathematica*, так и их эффективность при программировании различных приложений.

Литература

[1] Аладьев В.З., Хунт Ю.Я., Шишаков М.Л. Математика на персональном компьютере.– Беларусь: Гомель: BELGUT Press, 1996, 498 с., ISBN 34206140233

[2] Аладьев В.З., Ваганов В.А., Хунт Ю.Я., Шишаков М.Л. Введение в среду математического пакета *Maple V*.– Беларусь: Минск: Международная Академия Ноосферы, 1998, 452 с.

[3] Аладьев В.З., Ваганов В.А., Хунт Ю.Я., Шишаков М.Л. Программирование в среде математического пакета *Maple V*.– Минск–Москва: Российская Академия Экологии, 1999, 470 с., ISBN 4101212982

[4] Аладьев В.З., Ваганов В.А., Хунт Ю.Я., Шишаков М.Л. АРМ математика.– Таллинн–Минск–Москва: Российская Академия Естественных Наук, 1999, 608 с., ISBN 3420614023

[5] Aladjev V.Z. Computer Algebra Systems: A New Software Toolbox for *Maple*.– USA: Palo Alto: Fultus Corporation, 2004, ISBN 1596820004, 575 p.

[6] Аладьев В.З. Системы компьютерной алгебры. *Maple:* Искусство программирования.– Москва: БИНОМ, 2006, ISBN 5932081899, 792 с.

[7] Аладьев В.З., Бойко В.К., Ровба Е.А. Программирование и разработка приложений в системе *Maple*.– Гродно, Гродненский ун–т, 2007, ISBN 9789854178912, 456 с.

[8] Aladjev V.Z., Vaganov V.A. Modular programming: *Mathematica* vs *Maple*, and vice versa.- USA, CA: Palo Alto, Fultus Corporation, 2011, ISBN 9781596822689, 418 p.

[9] Аладьев В.З., Бойко В.К., Ровба Е.А. Программирование в системах *Mathematica* и *Maple:* Сравнительный аспект.– Гродно, Гродненский ун–т, 2011, ISBN 9789855154816.

[10] Аладьев В.З., Гринь Д.С. Расширение функциональной среды системы *Mathematica*.– Херсон: Изд–во Oldi–Plus, 2012, ISBN 9789662393729, 552 с.

[11] Аладьев В.З, Гринь Д.С., Ваганов В.А. Избранные системные задачи в *Mathematica*.– Херсон: Изд–во Oldi–Plus, 2013, ISBN 9789662890129, 556 с.

[12] Aladjev V.Z., Vaganov V.A. Extension of the *Mathematica* system functionality.– USA: Seattle, CreateSpace, An Amazon.com Company, 2015, ISBN 13: 9781514237823, 590 p.

[13] Aladjev V.Z. Library *UserLib6789* for system *Maple*.– the library can be freely downloaded from website https://yadi.sk/d/lVeHfqRDitLzw

[14] Aladjev V.Z. Package *AVZ_Package* for system *Mathematica*.– the package can be freely downloaded from website https://yadi.sk/d/JAveYeaIjJ9Ci

[15] Aladjev V.Z. Modular programming: *Maple* or *Mathematica* – A subjective standpoint / Int. school *«Mathematical and computer modeling of fundamental objects and phenomena in systems of computer mathematics»*, ed. Y.G. Ignat'ev.– Kazan: Kazan University Press, 2014, pp. 18–32.

[16] Аладьев В.З., Бойко В.К. Расширение функциональности систем *Mathematica* и *Maple*. Межд. школа-семинар по математическому моделированию в системах компьютерной математики *KAZCAS–2015*, под ред. Ю. Игнатьева.– Казань: Казанский ун-т, 2015, с. 95-112.

[17] Joyner D., William Stein W., et al. *Sage* Tutorial.– Lulu Press, 2011, 96 p.

[18] Кирсанов М.Н. *Maple 13* и *Maplet*. Решения задач механики.– М.: Лань, 2012, 503 с.

[19] Кирсанов М.Н. Практика программирования в системе *Maple*.– М.: Изд-во МЭИ, 2011

[20] Голоскоков Д.П. Практический курс математической физики в системе *Maple*.– СПб.: Изд–во ПаркКом, 2010, 644 с.

[21] Mangano S. *Mathematica* Cookbook.– CA: Sebastopol: O'Reilly Media, Inc., 2010, ISBN–13: 9780596520991, ISBN–10: 0596520999, 828 p.

[22] Gregor J., Tier J. Discovering Mathematics: A Problem–Solving Approach to Mathematical Analysis with *Mathematica* and *Maple*.– Springer–Verlag, 2010, 254 p.

Estonia, Tallinn, November, 2015