Программирование в системе *Mathematica.*

Работа с текстовыми данными и файлами.

Работа со строками

Хотя Mathematica ориентирована на математические приложения, в ней достаточно полно представлены функции для работы со строками (strings). Они могут потребоваться как для организации вывода текстовых сообщений (например надписей на графиках), так и для организации текстового диалога при разработке пакетов расширений и приложений системы. К тому же надо постоянно помнить, что Mathematica — система *символьной* математики, так что символьным преобразованиям, как сугубо математическим, так и общепринятым, в ней, естественно, уделено много внимания.

Многие функции для работы со строками выполняют общепринятые преобразования, имеющиеся в большинстве языков программирования высокого уровня. *Строкой* является произвольная цепочка символов, заключенная в кавычки, например "String". Ниже представлены некоторые функции для работы со строками:

- StringByteCount ["string"] — возвращает полное число байтов, используемых для хранения символов в строке "string";
- StringDrop ["string", {m, n}] — возвращает строку "string", удалив в ней символы от m до n;
- StringJoin["sl", "s2",...] или StringJoin [{ "s1", "s2",...}]-формирует строку, содержащую конкатенацию (объединение) указанных строк "s1";
- Stringlnsert ["string1", "string2",M] — вставляет строку "string2" в строку "string1", начиная с позиции M от начала этой строки (при отрицательном M позиция отсчитывается от конца указанной строки);
- StringLength ["string"] — возвращает число символов в строке;
- StringReplace["string", "s1 -> "spl"] или StringReplace["string", {"s1" -> "spl", "s2" -> "sp2",...}] — замещает "s1" на "spi" всякий раз, когда они появляются как подстроки "string";
- StringReverse ["string"] — меняет порядок символов в строке "string" на противоположный;
- StringPosition["string", "sub"] — возвращает список с позициями строки "sub" в строке "string" (дополнительные формы см. в справочной системе);
- StringTake ["string", n] — возвращает строку, состоящую из первых n символов строки "string";
- StringTake ["string", -n] — возвращает последние n символов из строки "string";
- StringTake ["string", {n}] — возвращает n-й символ в строке "string";
- StringTake ["string", {m, n}] — возвращает строку из символов, расположенных в позициях от m до n строки "string".

Эти функции хорошо известны программистам, работающим с современными языками программирования. Большое число дополнительных функций для работы со строками можно

найти в приложении. Обилие таких функций в языке программирования системы Mathematica указывает на его универсальный характер и обширные возможности в решении даже на первый взгляд далеких от математики задач. Ниже приведены примеры действия ряда функций работы со строками.

Таблица 1

Ввод (In)	**Вывод (Out)**
StringByteCount ["Hello ! "]	6
StringDrop ["Hello my friend!", 6]	my friend!
StringDrop ["Hello my friend! ", -10]	Hello
StringDrop ["Hello my friend! ", {7}]	Hello y friend!
StringDrop ["Hello my friend ! " , { 6 , 8 }]	Hello friend!
Stringlnsert ["Hello friend!"," my", 6]	Hello my friend!
StringJoin ["Hello"," my "]<>"friend!"	Hello my friend!
StringLength ["Hello"]	5
StringPosition["Hello my friend! ", "e"]	{{2, 2}, {13, 13}}
StringReplace["Hilo" , "i"->"el"]	Hello
StringReverse ["Hello ! "]	!olleH
StringTakef "Hello my friend!", 6]	Hello
StringTake["Hello my friend!", -8]	friend!
StringTake ["Hello my friend ! " , { 7 , 9 }]	my

Отметим еще несколько функций, относящихся к работе с символами и строками:

- FromCharacterCode [n] — возвращает строку, состоящую из одного символа с кодом n;
- FromCharacterCode [{n1, n2,...}] — возвращает строку, состоящую из последовательности символов с кодами ni;
- Characters ["string"] — возвращает список целочисленных кодов, соответствующих символам строки "string";
- ToLowerCase ["string"] — производит строку, в которой все буквы преобразованы в нижний регистр;
- ToString [expr] — возвращает строку, соответствующую форме вывода выражения expr. Опции устанавливают ширину линии, тип формата и т. д.;
- ToUpperCase ["string"] — вырабатывает строку, в которой все буквы преобразованы в верхний регистр;
- Unique [ ] — создает новый символ с именем в форме $nnn (nnn — уникальный порядковый номер);
- Unique [x] — создает новый символ с именем в форме x$nnn (nnn — уникальный порядковый номер);
- Unique [{x, y,...}]— создает список новых символов с уникальными именами;
- Unique ["xxx"] — создает новый символ с именем в форме xxxnnn (nnn — уникальный порядковый номер);

- Unique [name, {attrl, attr2,...}] — создает символ с указанными атрибутами attri;
- UpperCaseQ [string] — возвращает True, если все символы строки string являются прописными буквами (верхнего регистра), иначе возвращает False.

Примеры, приведенные ниже, показывают работу с этими функциями.

Ввод (In)	**Вывод (Out)**
ToCharacterCode ["Hello ! "]	{72,101,108,108,111,33}
FromCharacterCode [{72 , 101 , 108 , 108 , 111 , 33}]	Hello!
ToExpression ["2+3*4 "]	14
ToLowerCase ["HeLLo ! "]	hello!
ToUpperCase ["Hello"]	HELLO

Ввод (In)	**Вывод (Out)**
x:=ToString[2+3*4]	
X	14
Unique [ ]	$1
Unique [xyz]	xyz$2
Unique [xyz]	xyz$3
UpperCaseQ ["Hello"]	False
UpperCaseQ ["HELLO"]	True

Потоки и файлы

Система Mathematica имеет развитые средства для работы с потоками (streams) и файлами (files). Под *потоком* подразумевается непрерывная последовательность данных, циркулирующих внутри компьютера. Обмен потоками происходит практически непрерывно, например, при вводе поток ввода поступает от клавиатуры в компьютер, при печати поток данных поступает от компьютера в принтер через порт принтера и т. д.

Файлом является упорядоченная структура данных, имеющая имя и хранящаяся на каком-либо носителе, чаще всего на магнитном диске. Файлы могут иметь различные форматы и различный тип доступа к хранимой на них информации. Наиболее распространенные в системе Mathematica файлы документов являются файлами с последовательным доступом и имеют текстовый формат.

Последовательный доступ означает, что информация из открытого файла может быть считана строго последовательно от его начала до конца, отмеченного специальной меткой. Это напоминает считывание с магнитофонной кассеты. *Текстовый формат* означает, что все данные записаны в виде ASCII-кодов. Следовательно, прочесть такой файл можно с помощью любого текстового редактора, работающего с текстами в виде ASCII-кодов.

Потоки и файлы имеют много общего: имена, определенную структуру, необходимость открытия перед использованием и закрытия после использования. Однако если с файлами пользователь сталкивается уже в начале работы с системой (нужно вызвать файл с демонстрационным документом или сохранить его, а затем вызвать другой файл), то с понятием потока при работе с системой сталкиваться практически не приходится, хотя помимо нашей воли потоки данных постоянно текут между компьютером и его периферийным оборудованием.

Упрощенная работа с файлами

Прежде чем рассматривать весьма обширные возможности системы по работе с файлами в целом, отметим упрощенный прием вызова файла с помощью двойного символа «<<»:

```
<<filename
```

Эта команда считывает файл с указанным именем filename и зайосит в память компьютера содержащиеся в нем определения. Имя файла надо указывать полностью, то есть вместе с расширением. Исключением является случай, когда файл находится в основном каталоге системы. Эта команда эквивалентна функции

```
Get["filename", key]
```

Для записи объекта (переменной, массива, списка и т. д.) в файл служат упрощенные команды:

- expr >> filename — передает значение expr в файл с заданным именем;
- expr >>> filename — добавляет expr в конец файла с заданным именем.

Указанные команды по существу есть укороченные (и потому более удобные) формы следующих функций:

- Get ["filename", "key"] — читает файл, который закодирован функцией Encode с использованием ключа "key";
- GetContext ["context' "] — загружает файл с заданным контекстом;
- Put[exprl, expr2, ..., "filename"] — записывает последовательность выражений expri в файл с именем filename;
- PutAppend[expr1, expr2, ..., "filename"] — присоединяет последовательность выражений expri к файлу с именем filename.

Еще одна упрощенная функция — ! ! filename — выводит содержимое файла с заданным именем.

Следующие примеры показывают запись списка в файл C:\ma.vat, его считывание, затем добавление в файл еще одного списка и контроль контекста файла:

```
{{1,2,3},{4,5,6},{a,b,c}}>>C:\ma.val

<<C: \ma. val
```

```
{{1, 2, 3}, {4, 5, б), {a, b, c}} {d,e,f}>>>C: \ma.val

<<C: \ma. val

{d, e, f}

!!C:\ma.val

1, 2, 3, 4, 5, б, a, b, c d, e, f
```

Такая форма вызова особенно удобна для вызова файлов пакетов расширений и применений системы. Имя файла указывается по правилам, принятым в MS-DOS. Файлы пакетов применений имеют расширение .т. Мы уже приводили примеры использования определений, содержащихся в файлах пакетов расширения системы.

Имеется еще ряд функций для работы с файлами:

- ReadList ["filename"] — читает все оставшиеся в файле "filename" выражения и возвращает их в виде списка;
- ReadList ["filename", type] — читает из файла "filename" объекты указанного типа type до конца файла. Возвращает список считанных объектов;
- ReadList ["filename", {typel, type2,...}] — читает объекты указанных типов typei до конца файла filename;
- ReadList ["filename", types, n] — читает только первые n объектов указанных типов types из файла filename;
- Save ["filename", x1, x2,...] — создает файл с заданным именем filename, содержащий значения переменных x1, x2, ...;
- ! command — исполняет заданную команду операционной системы.

Допустим, что в каком-то текстовом редакторе создан файл с полным именем C:\datas.txt в ASCII-формате, содержащий просто шесть чисел с разделительными пробелами, размещенные в двух строках и представляющие массив 2x3 элемента:

```
1 11.2 34.5

2. 3.4 56
```

Тогда о структуре файла можно судить, используя команду

```
!!C:\datas.txt

1 1.2 34.5 2. 3.4 56.
```

Нетрудно заметить, что структура файла соответствует структуре массива. Однако считывание файла командой <<name дает следующий результат:

```
<<C: \datas. txt
```

```
380.8
```

Результат представляет вычисленное выражение второй строки файла. Считывание функцией ReadList без дополнительного аргумента также дает ошибочный результат:

```
ReadList["C:\datas.txt"]
```

```
{41.4, 380.8}
```

Нетрудно подметить, что функция восприняла каждую строку содержимого файла как результат перемножения трех чисел (пробел на языке Mathematica означает умножение). С дополнительным параметром Number все числа считываются верно:

```
ReadList["C:\datas.txt", Number]
```

```
{1, 1.2, 34.5, 2., 3.4, 56.}
```

Однако мы получили одномерный список — данные просто считываются построчно. Применение дополнительного параметра в виде {Number, Number} дает следующий результат:

```
ReadList["C:.txt", {Number, Number}]
```

```
{{1, 1.2), {34.5, 2.}, {3.4, 56.}}
```

Правильный результат можно получить, используя опцию RecordList->True: .

```
ReadList["C:.txt",Number,RecordLists->True]
```

```
{{1, 1.2, 34.5), {2., 3.4, 56.}}
```

Для загрузки файлов пакетов расширений (Add-On) используются функции, позволяющие задать контекст файлов:

- Needs ["context' ", "filename"] — загружает файл, если указанный контекст отсутствует в списке загруженных;
- Needs ["context^{s} "] — загружает файл, имя которого определяется с помощью функции ContextToFilename ["context^{ч} "], если указанный контекст отсутствует в списке загруженных.

Загрузка файлов с указанием их контекстов позволяет избежать конфликтов между разными пакетами расширения, используемыми одновременно.

Унифицированный подход

В общем случае для чтения данных из произвольного файла используются следующие функции:

- OpenRead[“*file*”] - открывает указанный файл для чтения и возвращает соответствующий объект типа InputStream.
- Read[*stream*] – считывает выражение из указанного потока ввода и возвращает его.
- Read[*stream*, *type*] – считывает один объект указанного типа и возвращает его.
- Read[*stream, {type1, type2, ...}*] – считывает и возвращает последовательность объектов указанных типов. В качестве типа считываемого объекта могут быть указаны:

Byte – один байт данных, представленный как целый код,

Character – одиночный символ,

Expression – законченное выражение Математики,

Number – целое или вещественное число в экспоненциальном формате,

Real - вещественное число в экспоненциальном формате,

Record – последовательность символов, разделённых строкой “\n”,

String – строка, завершающаяся символом новой строки,

Word – последовательность символов, разделённая либо строкой “\t”, либо “”.

Фунция Read возвращает значение EndOfFile, если достигнут конец файла.

- Close[*stream*] – закрывает указанный поток.

Пример ниже считывает первую строку из текстового файла и выводит её в окно ячейку ноутбука, если файл не пустой:

```
fileStream = OpenRead["C:\sampleFile.txt"];

textLine = Read[fileStream, String];

If[textLine ≠ "EndOfFile",

   Print[textLine];

   ];

Close[fileStream];
```

Чтение и запись текстов на русском языке

Поддержка русского языка реализована в Mathematica не полностью: вы можете набирать русский текст в ноутбуке, использовать русские символы в строковых константах, но при записи текста в файл или чтении файла любого формата (ASCII, UNICODE) русские буквы заменяются другими символами. В связи с этим, при необходимости чтения текста на русском языке из внешнего файла, его необходимо преобразовать в последовательность целых кодов, соответствующих русским буквам в Mathematica. Новый файл (с кодами) уже можно прочитать с помощью стандартных процедур Mathematica, а затем декодировать в строку. Для этого можно использовать следующие функции работы со строками:

- ToCharacterCode[“*string*”] – возвращает список целых кодов соответствующих символам строки *string*;
- FromCharacterCode[*n*] – возвращает строку, состоящую из символа с кодом *n*;
- FromCharacterCode[{*n*1, *n*2, …}] – созвращает строку, состоящую из последовательности символов с кодами *n*i;
- ToExpression[*input*] – возвращает выражение, полученное в результате интерпретации строки *input*.

Например, вот такой код

```
s = "Правила Дорожного Движения РФ";
clist = ToCharacterCode[s];
Print[clist];
```

выдаст список

```
{1055,1088,1072,1074,1080,1083,1072,32,1044,1086,1088,1086,1078,1085,1086,1075,1086,3
2,1044,1074,1080,1078,1077,1085,1080,1103,32,1056,1060}
```

Если этот список присвоить какой-то переменной как строку (в случае, если он считан из файла), то получить исходную строку можно так:

```
slist =
"{1055,1088,1072,1074,1080,1083,1072,32,1044,1086,1088,1086,1078,1085,1086,1075,1086,32,
1044,1074,1080,1078,1077,1085,1080,1103,32,1056,1060}";
clist = ToExpression[slist];
s = FromCharacterCode[clist];
Print[s];
```

При преобразовании русского текста в список кодов необходимо помнить, что перевод строки в Mathematica представлен всего одним кодом (10). Кавычки “ и перевод строки являются служебными сиволами, поэтому при поиске символа новой строки в тексте нужно искать подстроку “\n”, а при поиске кавычек – подстроку “\””.

Использование файлов других языков программирования

Из функций для работы с файлами особо надо отметить следующую функцию-директиву:

- Splice ["file .mx"] — вставляет в файлы на других языках программирования вычисленные выражения системы Mathematica, которые должны быть записаны в скобках вида <* и *>;
- Splice ["infile", "outfile"] — читает файл infile, интерпретирует фрагменты, содержащиеся между скобками <* и *>, и записывает результат в файл outfile.

Эта возможность особенно существенна при использовании программ на языках программирования C (расширение .me), Fortran (расширение .mf) и TeX (расширение .mtex), для

форматов которых Mathematica имеет средства конвертирования выражений (CForm, FortranForm и TexForm соответственно). Таким образом, имеется возможность экспорта выражений системы Mathematica в программы, составленные на этих языках.

Поясним применение функции-директивы Splice. Пусть имеется экспортированная программа на языке C, которая должна рассчитывать численное значение некоторого интеграла, и мы хотим получить формулу для этого интеграла средствами системы Mathematica. Допустим, она представлена файлом demo.me. Его можно просмотреть следующим образом:

!!demo.me

```
#include "mdefs.h"

double f(x)

double x;

{

double y;

y = <* Integrate[Sin[x]^5, x] *> ;

return (2*y- 1) ;

}
```

После исполнения функции Splice ["demo.me"] программа будет записана в файл demo.c, в котором выражение в скобках <*...*> заменено вычисленным значением интеграла (в форме CForm). Файл при этом будет выглядеть так:

!!demo.c

```
#include "mdefs.h" double f(x) double x;

{

double y;

y = -5*Cos(x)/8 + 5*Cos(3*x)/48- Cos(5*x)/80 ;

return (2*y- 1) ;

}
```

Запись определений

Из простых функций, обеспечивающих создание файлов с заданными определениями, надо отметить также функцию Save:

```
Save ["filename", symb1, symb2,...]
```

Она добавляет определения символов symbi к файлу filename (возможны упрощенные формы Save).

Приведем пример ее использования:

```
f[x_] = Sin[x] + y

y+ Sin[x]

y=a

a

Save["demol",f]

!!demol

f[x_] = y + Sin[x]

y = a
```

Другие функции для работы с файлами

В целом средства системы Mathematica обеспечивают возможности работы с различными файлами, присущие MS-DOS, без выхода из среды системы. Относящиеся к этой группе функции даны в приложении. Для этих функций характерно, что в момент выполнения они не дают видимого эффекта. К таким функциям относятся функции копирования директорий и файлов, смены их имен, удаления и т. д. Они хорошо известны пользователям MS-DOS и могут выполняться из среды Mathematica.

Рассматривая обширный список файловых и поточных операций, можно поневоле сделать вывод об их избыточности. Но здесь действует простое правило: не хочешь применять эти функции — не применяй! Они рассчитаны на пользователя, всерьез занимающегося стыковкой систем Mathematica с другими программными системами.

Важное место занимают функции, дающие информацию о директориях, файлах и потоках. К ним относятся следующие функции:

- Directory [ ] — возвращает текущий рабочий каталог;
- DirectoryStack [ ] — возвращает содержимое стека каталогов, которое представляет последовательность используемых в текущем сеансе каталогов;

- $Display— возвращает список файлов и каналов *(pipes*— канал или абстрактный файл), используемый функцией вывода $DisplayFunction по умолчанию;
- FileByteCount ["filename"] — возвращает количество байтов в файле;
- FileDate ["filename"] — возвращает дату и время последней модификации файла в виде списка;
- Filelnformation ["filename"] — возвращает информацию о файле;
- FileNames [ ] — приводит список всех файлов в текущем рабочем каталоге;
- FileNames [" form"] — перечисляет все файлы в текущем рабочем каталоге, чьи имена совпадают с шаблоном form;
- FileNames [{ "forml", "form2",...}] — перечисляет все файлы, чьи имена соответствуют любому из шаблонов formi;
- FileNames [forms, {"dirl", "dir2",...}] — перечисляет файлы с именами, соответствующими шаблонам forms, в любом из указанных каталогов diri;
- FileType ["filename"] — возвращает тип файла: File, Directory или None (если указанного файла не существует);
- $HomeDirectory — дает имя «домашней» директории пользователя;
- $Output — дает список файлов и каналов, в которые направляется стандартный вывод системы Mathematica;
- ParentDirectory [ ] — возвращает имя родительского каталога для текущего рабочего каталога;
- ParentDirectory ["dir"] — возвращает имя родительского каталога для каталога dir;
- $ Path — дает список каталогов для просмотра при попытке поиска внешнего файла;
- StreamPosition [stream] — возвращает целое число, которое указывает позицию текущей точки в открытом потоке stream;
- Streams [ ] — возвращает список всех потоков, открытых в данный момент;
- Streams ["name"] — перечисляет только потоки с указанным именем name.

Приведенные ниже примеры иллюстрируют использование большинства из этих достаточно простых функций:

```
Directory[]

C:\PROGRAM FILES\WOLFRAM RESEARCH\MATHEMATICA\4.0

DirectoryStack[]

{} / $Display

stdout

FileByteCount["C:.val"]

46

FileDatef'C: .val"]
```

```
{1999, 8, 3, 16, 4, 44}
```

FileInformation["C:.val"]

```
{File->C:\ma.val, FileType->File, Date -> 3142685084, ByteCount ->46}
```

Filenames[]

```
{Examples, FILES, MATHEMATICA.EXE,

MATH.EXE, MATHINSTALLER.EXE, MATHKERNEL.EXE}
```

FileType["C:.val"]

File HomeDirectory[]

c:\ $0utput

```
{OutputStream[stdout, 1]}
```

ParentDirectory[]

C: \m3 Streams[]

```
{OutputStream[stdout, 1],

OutputStream[stderr, 2]}
```

Высказанное выше соображение об избыточности набора операций вполне применимо и для этих функций.