Chapter 12
Input–Output and Strings

12.1 Reading and Writing .m Files

When developing any nontrivial *Mathematica* program, it is better to write it in a text file, using a text editor, and then to read it into a fresh *Mathematica* session. In this way, your actions will be reproducible. You can fix a bug and see what has changed. Suppose we have a text file called wrong.m. It contains the text

```
In[1] := FilePrint["wrong.m"]
a=x^2+2*x*y+y^2;
b=x^3+3*x^2*y
+3*x*y^2+y^3;
```

We can read it. Now the variables *a* and *b* have values:

```
In[2] := <<wrong.m
In[3] := {a,b}
```

Out[3] $= \{x^2+2xy+y^2, x^3+3x^2y\}$

The value of *b* is not what we expected. Why? When *Mathematica* sees an end of a line, it checks if the text read so far forms a syntactically correct expression. If so, the expression is considered complete; the next line starts a new expression. Otherwise, the next line is considered a continuation of the current expression. Thus our file wrong.m contains not two but three expressions: two assignments (to *a* and *b*) and a separate polynomial consisting of two terms. One way to prevent such unintended splitting of multiline expressions is to place a binary operator (e.g., $+$ or $-$) at the end of each line. *Mathematica* always writes its result in such a way. But it is easy to forget about it when writing a long expression in a text editor. Also, if we paste results from some other system into a *Mathematica* program, it would be tedious and error-prone to bring long expressions to such a form by hand. It is better to enclose each multiline expression in extra parentheses, then any incomplete subset of lines is not syntactically correct. Therefore we edit our wrong.m and obtain a file right.m:

In[4] := FilePrint["right.m"]

```
a=x^2+2*x*y+y^2;
b=(x^3+3*x^2*y
+3*x*y^2+y^3);
```

Now everything's all right:

In[5] := <<right.m

In[6] := {*a*,*b*}

Out[6] = $\{x^2 + 2xy + y^2, x^3 + 3x^2y + 3xy^2 + y^3\}$

In[7] := Clear[*a*,*b*]

Let us modify the file:

In[8] := FilePrint["right2.m"]

```
a=x^2+2*x*y+y^2;
(x^3+3*x^2*y
+3*x*y^2+y^3)
```

Now the last expression is not an assignment, but just a polynomial. What happens if we read it?

In[9] := <<right2.m

Out[9] = $x^3 + 3x^2y + 3xy^2 + y^3$

In[10] := *a*

Out[10] = $x^2 + 2xy + y^2$

The operator << (its full name is Get) returns the value of the last expression. Therefore, we can use it in an assignment (or as an argument of any other function):

In[11] := *b* = <<right2.m

Out[11] = $x^3 + 3x^2y + 3xy^2 + y^3$

In[12] := {*a*,*b*}

Out[12] = $\{x^2 + 2xy + y^2, x^3 + 3x^2y + 3xy^2 + y^3\}$

In[13] := Clear[*b*]

The function Get tries to find the file in all directories in the list $Path. Its default value contains, in particular, the current directory ".". You can add more directories to it using list operations, such as Append, Prepend, or Join.

And this operator (its full name is Put) writes the value of an expression into a file. The value is written in the input form, and hence can be later read by *Mathematica*.

In[14] := *a*>>result.m

In[15] := FilePrint["result.m"]

```
x^2 + 2*x*y + y^2
```

In[16] := Clear[*a*]

In[17] := *a* = <<result.m

Out[17] = $x^2 + 2xy + y^2$

In[18] := *a*

Out[18] = $x^2 + 2xy + y^2$

The function >> writes just an expression, not an assignment to some variable. Often it is more useful to write an assignment (or several of them) which defines some variable (or function). Suppose we have

In[19] := *f*[0] = 1; *f*[*n*_] := *n* ∗ *f*[*n* − 1]

We can save the definition of the function f into a file:

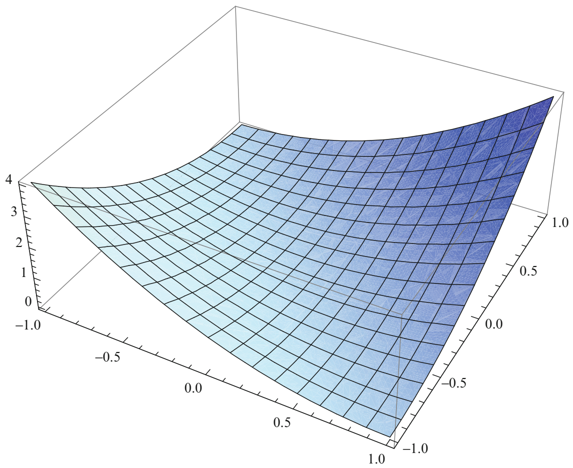
In[20] := Save["f.m", *f*]
In[21] := FilePrint["f.m"]
f[0] = 1
$f[n\_] := n * f[n-1]$
In[22] := Clear[*f*]
In[23] := <<f.m
In[24] := *f*[10]
Out[24] = 3628800
In[25] := Clear[*f*]

12.2 Output

The function Print prints expressions (including strings, plots, etc.). It does not separate them by spaces, so that it is usually a good idea to insert " " between expressions. It adds a newline at the end.

In[26] := *s* = "A strings\nwith a newline";
In[27] := *p* = Plot3D[*a*, {*x*, −1, 1}, {*y*, −1, 1}];
In[28] := Print[*s*, " ", *a*, " ", *p*]
Out[28] = A strings

with a newline $x^2 + 2xy + y^2$



In[29] := Clear[*p*]

The function Print is most useful when you want to know what happens at some point deep inside your own function when it is called. If you want to write expressions to a file instead, use Write:

In[30] := *f* = OpenWrite["res.m"]
Out[30] = OutputStream[res.m, 19]
In[31] := Write[*f*, *s*]; Write[*f*, *a*]
In[32] := Close[*f*]
Out[32] = res.m
In[33] := FilePrint["res.m"]

A strings\nwith a newline
x^2 + 2*x*y + y^2
Expressions are written in the input form and can be read later.
In[34] := Clear[*s*, *a*]

12.3 C, Fortran, and TEX Forms

Suppose you have derived analytically a valuable expression *a*:
In[35] := *a* = Sin[*x*]^2/*x*^2 − 1
Out[35] $= -1 + \frac{\mathrm{Sin}[x]^2}{x^2}$
Now you want to do some large-scale numerical calculations with it. In order to avoid possible errors when translating this expression into a form suitable to inclusion into a numerical program, it is a good idea to do this step automatically. The function CForm converts an expression into a form which can be inserted into a C (or C++) program.
In[36] := CForm[*a*]
Out[36] = -1 + Power(Sin(x),2)/Power(x,2)
The macros Power, Sin, and friends are defined in the file mdefs.h which is supplied with *Mathematica*. If the expression contains special functions, you will need some extra C libraries which can calculate them numerically. Of course, it is better to write the C form of an expression into a file and then insert it into your program. For those old-fashioned enough to do numerical computations in Fortran, there is also FortranForm.
In[37] := FortranForm[*a*]
Out[37] = -1 + Sin(x)**2/x**2

At last, all computations are done, and you are writing an article to be submitted to a scientific journal. You want to include your valuable expression *a* and to avoid errors in process of doing so. Scientific articles are written in LATEX (in most cases). The function TeXForm converts an expression into a form which can be inserted into a LATEX file.
In[38] := TeXForm[*a*]
Out[38] = \frac{\sin ^2(x)}{x^2}-1
In[39] := Clear[*a*]

12.4 Strings

The simplest string operation is concatenation:
In[40] := s1 = ”This”; s2 = ”is” ; s3 = ”a string”;
In[41] := *s* = s1 <> ” ” <> s2 <> ” ” <> s3
Out[41] = This is a string

In[42] := FullForm[*s*]
Out[42]//FullForm =
    ”This is a string”
In[43] := StringLength[*s*]
Out[43] = 16
In[44] := Clear[*s*, s1, s2, s3]

Mathematica has string patterns and substitutions.

In[45] := StringReplace[”abcabd”, ”ab” → ”AB”]
Out[45] = ABcABd

$x\_$ means an arbitrary (single) character.

In[46] := StringReplace[”a1b a2b a3c”, a ~~ *x*_ ~~ b → A ~~ *x* ~~ B]
Out[46] = A1B A2B a3c
In[47] := StringReplace[”a1b2 a12b”,
    a ~~ *x*_ ~~ b ~~ *y*_ → A ~~ *y* ~~ B ~~ *x*]
Out[47] = A2B1 a12b

Internally, these patterns are the function StringExpression.

In[48] := FullForm[a ~~ *x*_]
Out[48]//FullForm =
    StringExpression[a, Pattern[*x*, Blank[]]]

A pattern with two identical arbitrary characters.

In[49] := StringReplace[”a1b1 a1b2 a2b2”,
    a ~~ *x*_ ~~ b ~~ *x*_ → A ~~ *x* ~~ *x* ~~ B]
Out[49] = A11B a1b2 A22B

b|c means b or c.

In[50] := StringReplace[”abcd abcd”, b|c → X]
Out[50] = aXXd aXXd

$x\_\_$ means any nonempty sequence of characters.

In[51] := StringReplace[”ab”, a ~~ *x*__ ~~ b ~~ *x*__ → A ~~ *x* ~~ B ~~ *x*]
Out[51] = ab
In[52] := StringReplace[”a12b12”,
    a ~~ *x*__ ~~ b ~~ *x*__ → A ~~ *x* ~~ B ~~ *x*]
Out[52] = A12B12

$x\_\_\_$ means any sequence of characters (including empty).

In[53] := StringReplace[”ab”, a ~~ *x*___ ~~ b ~~ *x*___ → A ~~ *x* ~~ B ~~ *x*]
Out[53] = AB
In[54] := StringReplace[”a12b12”,
    a ~~ *x*___ ~~ b ~~ *x*___ → A ~~ *x* ~~ B ~~ *x*]
Out[54] = A12B12

As in the case of general patterns, /; means a condition (such that).

In[55] := StringReplace[”a1b1 a1b2”,
    a ~~ *x*_ ~~ b ~~ *y*_/;*x*!=*y* → A ~~ *x* ~~ *y* ~~ B]
Out[55] = a1b1 A12B

The function StringMatchQ tests if a string matches a pattern.

```
In[56] := {StringMatchQ["a1b1", a ~~ x_ ~~ b ~~ x_],
      StringMatchQ["a1b2", a ~~ x_ ~~ b ~~ x_],
      StringMatchQ["a1b1c", a ~~ x_ ~~ b ~~ x_]}
Out[56] = {True, False, False}
```

The function StringFreeQ tests if there are no substrings matching a pattern.

```
In[57] := {StringFreeQ["a1b1", a ~~ x_ ~~ b ~~ x_],
      StringFreeQ["a1b2", a ~~ x_ ~~ b ~~ x_],
      StringFreeQ["a1b1c", a ~~ x_ ~~ b ~~ x_]}
Out[57] = {False, True, False}
```

The function StringSplit splits a string at each occurrence of a pattern; if the second argument is not given, then at each white space.

```
In[58] := StringSplit["xxa1byya2bzz", a ~~ x_ ~~ b]
Out[58] = {xx, yy, zz}
In[59] := StringSplit["xx yy zz\nuu\tvv\t\t ww"]
Out[59] = {xx, yy, zz, uu, vv, ww}
```

Mathematica contains many more string manipulation functions and additional powerful features of string patterns and can be used for text processing instead of Perl. For more details, see the online help.