Introduction to Maple Programming

Example Maple procedure that operates on two lists

```
> #-------------------------------------------------------------
  # ladd: Returns list lres[i] such that lres[i] = l1[i] + l2[i]
  #-------------------------------------------------------------
  ladd := proc(l1::list(algebraic), l2::list(algebraic))
>
>    # Declare local vbl for building up SEQUENCE of values
>    # that will then be converted to a list and returned,
>    # and a loop vbl.
>
>    local sres, i;
>
>    # Check that the lists have the same length.  If not, invoke
>    # error to print message and exit.
>
>    if nops(l1) <> nops(l2) then
>       error("Input lists l1 and l2 are not of the same length");
>    end if;
>
>    # NULL is a special value that evaluates to a null (empty)
>    # expression sequence.
>
>    sres := NULL;
>
>    # Loop over all of the elements in l1, l2 ...
>
>    for i from 1 to nops(l1) do
>
>       # Build up the sequence.
>
>       sres := sres , (l1[i] + l2[i]);
>
>    end do;
>
>    # Convert the built up sequence to a list and return the
>    # list using Maple's default mechanism for returning a value
>    # from a procedure.
>
>    [sres]
>
```

```
> end proc;
```

ladd := **proc**(*l1*::list(*algebraic*), *l2*::list(*algebraic*))
local *sres*, *i*;
if nops(*l1*) ≠ nops(*l2*) **then error** "Input lists l1 and l2 are not of the same length" **end if**;
sres := *NULL*;
for *i* **to** nops(*l1*) **do** *sres* := *sres*, *l1*[*i*] + *l2*[*i*] **end do**;
[*sres*]
end proc

Usage examples: valid input

```
> list1 := [2, 4*x^2, 3.0, 6*cos(y)^2];
```

$$list1 := [2, 4\,x^2, 3.0, 6\cos(y)^2]$$

```
> list2 := [z, 2*x^2, 5, 6*sin(y)^2];
```

$$list2 := [z, 2\,x^2, 5, 6\sin(y)^2]$$

```
> ladd(list1, list2);
```

$$[2+z, 6\,x^2, 8.0, 6\cos(y)^2 + 6\sin(y)^2]$$

How could we modify `ladd` so that the fourth element in the above list was simplified to 6?

```
> ladd([], []);
```

$$[\;]$$

```
> ladd(list1, list1);
```

$$[4, 8\,x^2, 6.0, 12\cos(y)^2]$$

```
> ladd(list1, [1, 2, 3, 4]);
```

$$[3, 4\,x^2+2, 6.0, 6\cos(y)^2+4]$$

Usage examples: invalid input

```
> list3 := [a, b, c];
```

$$list3 := [a, b, c]$$

```
> list4 := [a, b, c, [d, e]];
```

$$list4 := [a, b, c, [d, e]]$$

```
> ladd(list1,list3);
Error, (in ladd) Input lists l1 and l2 are not of the same length
```

```
> ladd(list1,list4);
Error, invalid input: ladd expects its 2nd argument, l2, to be of type
list(algebraic), but received [a, b, c, [d, e]]
```

Illustration of Maple's built in tracing (debugging) facility

See ?trace for full details.

Enable tracing of procedure `ladd`

```
> trace(ladd);
```

$$ladd$$

```
> ladd(list1, list2);
{--> enter ladd, args = [2, 4*x^2, 3.0, 6*cos(y)^2], [z, 2*x^2, 5, 6*sin(y)^2]
```

$$sres :=$$

$$sres := 2 + z$$

$$sres := 2 + z, 6\,x^2$$

$$sres := 2 + z, 6\,x^2, 8.0$$

$$sres := 2 + z, 6\,x^2, 8.0, 6\cos(y)^2 + 6\sin(y)^2$$

$$[2 + z, 6\,x^2, 8.0, 6\cos(y)^2 + 6\sin(y)^2]$$

```
<-- exit ladd (now at top level) = [2+z, 6*x^2, 8.0, 6*cos(y)^2+6*sin(y)^2]}
```

$$[2 + z, 6\,x^2, 8.0, 6\cos(y)^2 + 6\sin(y)^2]$$

Note that in addition to the input arguments, and output return value, the tracing output includes the return value of all of the statement body that were terminated with a semi-colon (which is all of the statements).

Had any of the statements been terminated with a colon, they would *not* have been traced, so if you want to use the **trace** facility to full advantage, be sure to use semi-colons rather than colons when coding procedures.

Disable tracing of the procedure

```
> untrace(ladd);
```

$$ladd$$

Using the op command to display the definition of a procedure

Example:

```
> op(ladd);
```

proc($l1$::list($algebraic$), $l2$::list($algebraic$))
local $sres$, i;
  if nops($l1$) $\neq$ nops($l2$) **then error** "Input lists l1 and l2 are not of the same length" **end if**;
  $sres$:= $NULL$;
  for i **to** nops($l1$) **do** $sres$:= $sres$, $l1[i] + l2[i]$ **end do**;
  [$sres$]
end proc

By default, **op** will *not* display the full definition of most standard Maple commands/procedures

```
> op(sin);
```

proc(x::$algebraic$) ... **end proc**

```
> op(diff);
```

proc() **option** $builtin = diff$, $remember$; **end proc**

The interface command

Quoting from the help page for interface: ... this function is used to set and query all variables that affect the format of the output, but do not affect the computation.

General syntax:

`interface(<name>=<value>)`

or

`interface(<name>)`

Example 1: Viewing the definition of standard Maple commands

```
> interface(verboseproc=2);
```

1

Note that interface always returns the old value of the interface variable that is being set; i.e. **`verboseproc = 1`** is the default (initial setting)

```
> op(sin);
```

$$\mathbf{proc}(x::algebraic)\ \ \ldots\ \ \mathbf{end\ proc}$$

The **`diff`** command is a "**builtin**", meaning that it is *not* coded in Maple, but rather in the implementation language for the Maple system itself (which is C).

Thus, the output from **`op(diff)`** is still not very revealing

```
> op(diff);
```

$$\mathbf{proc}()\ \mathbf{option}\ builtin = diff, remember;\quad \mathbf{end\ proc}$$

Example 2: Changing the way Maple outputs expressions (useful for Homework 2, Problem 5)

Setting **`prettyprint=0`** produces "flat" (i.e. 1-dimensional) output

```
> interface(prettyprint=0);
3
```

```
> (x+y)^3;
(x+y)^3
```

```
> exp(2000.0);
.3881180194e869
```

Restore the default value

```
> interface(prettyprint=3);
```

$$0$$

```
> (x+y)^3;
```

$$(x+y)^3$$

```
> exp(2000.0);
```

$$0.3881180194\ 10^{869}$$

The sprintf function (useful for Homework 2, Problem 4)

See ?sprintf for full details: this function will be familar to you if you know some C (or C++)

Usage of sprintf is best illustrated by way of an example

```
> expr1 := 2*cos(x)*sin(x);
```

$$expr1 := 2\cos(x)\sin(x)$$

The first argument to **`sprintf`** is a string in which "formatting specifications", which begin with a **%**, are embedded. Each additional argument to **`sprintf`** should be a Maple expression, and for each extra argument, there should be a corresponding formatting specification.

Each of the supplied Maple expressions is then converted to a string according to its formatting specification, and the resulting string replaces the the formatting specification in the string that is returned by **`sprintf`**. (Confused yet?)

For our purposes, the most useful formatting specification for use with **`sprintf`** is **%a**, which will convert *any* Maple expression into a string, as the following example illustrates.

```
> string1 := sprintf("The value of expr1 is %a", expr1);
```

string1 := "The value of expr1 is 2*cos(x)*sin(x)"

Note that the string returned by **`sprintf`** can then be manipulated using any/all of the facilities that Maple provides for string processing. For example, we can use the **`cat`** command to prepend another string to **`string1`**

```
> cat("I've said it before: ",string1);
```

"I've said it before: The value of expr1 is 2*cos(x)*sin(x)"

Although the **%a** format specification will suffice for the purposes of Homework 2, should you want to use Maple to display and/or output numerical data, it is well worth learning about some of the other format specifications such as **%d** for integers and **%f**, **%e** and **%g** for floating point values.

We will return to this in this afternoon's lab where we will code a Maple procedure that writes some output to a file.

Finally, note that we could also have converted **expr1** to a string using the **convert** command:

```
> convert(expr1,string);
```

"2*cos(x)*sin(x)"

However, I introduced the **`sprintf`** command here, and recommend its use

in Homework 2, in part due to its intimate relationship to the commands
1) `printf`, which gives you much more control over the appearance of Maple output, especially for numeric values, than that provided by **`print`** and
2) `fprintf`, which you *need* to use should you want to perform output to a file from Maple.