

Set Theory - Maple Programming Assignment

Mth 355 Fall 2001 Assignment 2. Due Oct 12.

Mth 355/399 Oct 5 2001 Maple 6

Bent E. Petersen

Filename: 355f2001-assign-002-maple-prog.mws

Now that you have had a bit of experience with Maple in the MLC lab, you are probably ready to write some code of your own. Don't forget the on-line help.

Procedures in Maple can have any number of arguments, specified or not. Maple uses nargs to tell a procedure how many arguments it was called with. The individual arguments are accessed through the args[] array. Here is an example, mintersect(), which will compute the intersection of any finite number of sets (so multiple intersect):

```
> mintersect:=proc()
> local k,Z;
> if nargs = 0 then
>   Z:= {}; # empty intersection is empty
> elif nargs = 1 then
>   Z:= args[1]; # nothing else to do
> else
>   Z:= args[1];
>   for k from 2 to nargs do
>     Z:= Z intersect args[k];
>   od;
> fi;
> return Z;
> end;
```

Note the local variables are private to the procedure. Even if you have assigned a value to k outside the procedure above it will run correctly, and it will not disturb the value assigned to k.

Let's test it:

```
> mintersect();
{ }
> A1:={1,2,4,6,7,5,9,8}; A2:={2,5,7,8,1}; A3:={2,5,8,3};
A4:={1,3,5,9,4}; A5:={3,2,6}; A6:={2,3,6,7,8};
A1 := {1, 2, 4, 5, 6, 7, 8, 9}
A2 := {1, 2, 5, 7, 8}
```

```
A3 := {2, 3, 5, 8}
A4 := {1, 3, 4, 5, 9}
A5 := {2, 3, 6}
A6 := {2, 3, 6, 7, 8}
```

```
> mintersect(A1,A2,A3);
{2, 5, 8}
> mintersect(A1,A2,A3,A4);
{5}
> mintersect(A1,A2,A3,A4,A5);
{ }
```

Looks OK!

Problem 1. Write a Maple procedure, `munion()`, to compute arbitrary unions. Be sure to check it works correctly.

Here's another example where one does not want to specify the number of arguments *a priori*.

Problem 2. Write a procedure, `mmax()`, to compute the maximum value of an arbitrary finite number of real numbers.

Recall our procedure, `carp()`, for computing the Cartesian product of two sets:

```
> carp:=proc(X,Y)
>   local Z,x,y;
>   Z:={};
>   for x in X do
>     for y in Y do
>       Z:=Z union {[x,y]};
>     od;
>   od;
>   return Z;
> end;
```

Thus

```
> carp(A3,A4);
{[5, 9], [8, 1], [8, 3], [8, 4], [8, 5], [3, 1], [3, 3], [3, 4], [3, 9], [5, 1], [5, 3], [5, 4], [5, 5],
 [2, 1], [2, 3], [2, 4], [2, 5], [2, 9], [3, 5], [8, 9]}
```

For the Cartesian product of A3, A4 and A5 we have

```

> Z:={}: for x in carp(A3,A4) do for a in A5 do Z:=Z union
  {[op(x),a]}; od; od;
> Z;
{[3, 3, 2], [3, 3, 3], [3, 3, 6], [3, 5, 2], [3, 5, 3], [3, 5, 6], [3, 9, 2], [3, 9, 3], [3, 9, 6],
  [5, 3, 2], [5, 3, 3], [5, 3, 6], [5, 5, 2], [5, 5, 3], [5, 5, 6], [5, 9, 2], [5, 9, 3], [5, 9, 6], [3, 4, 2],
  [3, 4, 3], [3, 4, 6], [5, 4, 2], [5, 4, 3], [5, 4, 6], [2, 1, 2], [2, 1, 3], [2, 1, 6], [2, 3, 2], [2, 3, 3],
  [2, 3, 6], [2, 4, 2], [2, 4, 3], [2, 4, 6], [2, 5, 2], [2, 5, 3], [2, 5, 6], [2, 9, 2], [2, 9, 3], [2, 9, 6],
  [3, 1, 2], [3, 1, 3], [3, 1, 6], [5, 1, 2], [5, 1, 3], [5, 1, 6], [8, 1, 2], [8, 1, 3], [8, 1, 6], [8, 3, 2],
  [8, 3, 3], [8, 3, 6], [8, 4, 2], [8, 4, 3], [8, 4, 6], [8, 5, 2], [8, 5, 3], [8, 5, 6], [8, 9, 2], [8, 9, 3],
  [8, 9, 6]}

```

Note `op(x)` is the list of entries in the list `x`. Thus

```

> op([3,4]);
          3, 4

```

Problem 3. Write a Maple procedure, recursive if you wish, to compute the Cartesian product of an arbitrary finite number of sets.

In Problem 3 it is probably safer to make use of `nargs` and `args[]` rather than using recursion.

Again perform a check (using the sets above if you wish). You might also try the case when one factor is empty - you should get the empty set.

The Boolean values `true` and `false` are keywords in Maple. You can assign them to variables directly or as result of Boolean operations. As an example recall our subset test

```

> subset:=proc(X,Y)
>   local x,s;
>   s:=true;
>   for x in X do
>     s:= s and member(x,Y);
>   od;
> end:
>
> subset(A3,A4);
          false
> subset(A5,A6);
          true

```

Problem 4. Write a Maple procedure which accepts as input any finite number of finite sets and returns true if they are pairwise disjoint and returns false otherwise.

Test your procedure with B1, B2, B3, B4 and also with B1, B2, B3, B4, B5 where

```
> B1:={2,3}; B2:={4,7}; B3:={5,8}; B4:={1,6,9}; B5:={7,9};  
      B1 := {2, 3}  
      B2 := {4, 7}  
      B3 := {5, 8}  
      B4 := {1, 6, 9}  
      B5 := {7, 9}
```

Note sets need not be defined by tediously listing all the elements. Here's an example using the seq() command:

```
> {seq(n^2,n=1..20)};  
      {1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}
```

Here's another example which uses a do-loop:

```
> A:={}: for k from 1 to 50 do A:= A union {ithprime(k)}; od;  
> A;  
      {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101,  
      103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,  
      199, 211, 223, 227, 229}
```

Finally here is an (trivial) example where a set is specified as the set of solutions of an equation:

```
> p:=(t-1)*(t-2)*(t-3)*(t-4);  
      p := (t - 1) (t - 2) (t - 3) (t - 4)  
> B:={solve(p=0,t)};  
      B := {1, 2, 3, 4}
```

The Maple function call irem(m,n) returns the integer remainder when m is divided by n. Here are some examples

```
> irem(75,12); irem(123,29);  
      3  
      7
```

Problem 5. Write a procedure `g(a,b)` which returns the set consisting of all integers from `a` to `b` which are divisible by 3 and by 5, but are not divisible by 7.

Your code should probably start something like

```
g:=proc(a,b)
local A,B,...;
if nargs<>2 then
  return FAIL;
fi;
A:=ceil(a); B:=floor(b);
.....
end:
```

Have fun!

>