Programming in Maple

Dylan Zwick

Monday, June 23rd, 2008

Today we're going to learn the basics of Maple programming on a "need to know" basis. In other words, we'll learn the bare bones basics of Maple programming in a similar fashion to how we learned the bare bones basics of programming in C++. However, we'll be even more cursory in our treatment than we were with C++, because you've now all had some programming experience.

The goal is that by the end of the day you'll have seen enough Maple programs, and you'll understand the concepts sufficiently, that you can write a program that finds the GCD of two polynomials in the variable x. If you finish this quickly and you want to go on (and you know a little about recursion) you can then construct a program that finds a single generator for the ideal generated by a set of polynomials in x.

Note - We know this can always be done with just one polynomial as the ring of polynomials in one variable is a Euclidean domain, and is therefore a principal ideal domain.

1 Creating Procedures in Maple

The basic format for creating a procedure in mapel is:

```
NAME := proc(variables) [local variables;]
statements;
end;
```

This creates a procedure with the name of NAME. So, for example, here is a procedure that takes a variable and returns that variable plus 1.

```
AddOne := proc(x)
x+1;
end;
```

If you enter this into Maple and then type in:

AddOne(2);

Maple will return 3. Sweet. However, if on the other hand you type in:

AddOne(x);

Maple will return x+1. Maple has built in functionality that makes it much more flexible when it comes to the types of objects you're using. That's one of the reasons we're using Maple now instead of C++.

Now, just as with C++ the two most important commands we'll have are while loops and if statements. The if statement in Maple is similar to the if statement in C++. Here is a procedure that illustrates the if statement.

```
IsEven := proc(x)
if not type(x,numeric) then ERROR(‘Must Input a Number‘) fi;
if irem(x,2) = 0 then true
else false
fi;
end;
```

We can see a few things by examining even this simple procedure. First, we see that we signal the end of an "if" statement with a "fi". Get it? Also, we've introduced two new functions: type and irem. The first of these takes two arguments, the first is whatever variable you're looking at, and the second is a type of variable. The type function returns true if the input to the first argument is of the type listed in the second argument, and it returns false otherwise. The irem function returns the integer remainder of the first argument upon division by the second. It's basically the mod operator from C++.

Now, as an example of a while loop, let's write a procedure that divides a number by 2 until it can't do it any more. So, for example, running this procedure on 16 would return 1, while running it on 20 would return 5.

```
Strip2 := proc(x) local p;
p := x;
while IsEven(p) do p := p/2 od;
p;
end;
```

Now, we note two things here. First, a "while" statement is followed by whatever must be satisfied in order for it to proceed, and then followed by a "do" statement which tells the procedure what to do. The do statements ends with "od". We also note two other things about this procedure. First, we used our IsEven program in its implementation, which is fine. However, if we wrote this without writing IsEven first, Strip2 wouldn't run. Second, we created the local variable p. The reason we did this is because if we had included the statement $x := x/2$ Maple would have complained. It does not like it when you redefine an argument that is sent to the function.

2 Implementing Polynomial Division

First, type in the following procedure. See if you can reason out what it's doing, but if not don't worry about it yet.

```
TERMDEGREE := proc(f)
if type(f,numeric) then 0
```

```
elif type(f,name) then 1
elif type(f,'*') then TERMDEGREE(op(2,f))
elif type(f,'^') then op(2,f)
else ERROR('Not a recognized monomial')
fi;
end;
```

This procedure will give the degree of a term in a polynomial. So, for example, TERMDEGREE($4x^3$) will return 3.

Now, we're going to use this to create a function that returns the leading term of a polynomial. Such a program is:

```
LT := proc(f) local i; i:= 1;
if not type(f,polynom) then ERROR('Wrong type of input.') fi;
if type(f,'+') then while TERMDEGREE(op(i,f)) <> degree(f,x)
do i := i+1 od;
op(i,f);
else f;
fi;
end;
```

Now, it is a good idea to deconstruct this and figure out what's going on. First, the procedure uses the "op" function. Any object in Maple is treated as a directed acyclic graph. It's not important that you know what that is, it's just important that you understand that the op function is used to pick objects apart. So, for example, if $f = x^2 + 3x - 2$ Maple stores this as the sum of three other objects. So $op(2, f)$ would return the second term in this sum, namely, $3x$. Now, $3x$ itself is stored as the product of two other objects. So, the command $op(1, 3x)$ would return 3, and you can nest these to get $op(1, op(2, f))$, which would again evaluate to 3. To find out the number of operands an object has use the command $nops(f)$. In our example $nops(f) = 3$. Second, we used the built in Maple command "degree". The "degree" function takes as its input a polynomial and a variable, and it returns the degree of that variable in the polynomial. The reason we have to go through all this trouble to write the leading term command (as compared to just using, for example, $op(1, f)$) is that we want to not have to worry about the order in which the terms of the polynomial are presented. We want the leading term of $x^2 + 2x - 1$ to evaluate to the same thing as the leading term of $2x + x^2 - 1$. Which, of course, it should.

Finally, here is a procedure that takes in two polynomials, g, f, as input and writes them as:

$$f = qg + r$$

where q and r are other polynomials and $deg(r) < deg(f)$.

```
POLYDIV := proc(g,f) local q,r;
q := 0;
r := f;
while r <> 0 and divide(LT(r),LT(g))
do
q := collect(q + LT(r)/LT(g),x);
r := collect(r - (LT(r)/LT(g))*g,x);
od;
[q,r];
end;
```

Three things about this procedure. We note first that it uses the "divide" function. This is a function built into Maple that determines if a given polynomial divides another polynomial. Second, the collect command makes sure that the polynomial is simplified. So, for example, $x(x^4 - 2)$ evaluates to $x^5 - 2x$. Finally, the output of the procedure is a *list*. It is a list of two elements; namely, the polynomials q and r. If we wanted to simply return the remainder, for example, we would write $POLYDIV(g, f)[2]$, which would return the second element in the list.

3 Now It's Your Turn

OK, using what we've learned so far and the functions that we've written you now should be able to write a procedure that takes as its input two polynomials, f and g, and returns their greatest commond divisor. Write a procedure that does this, and call it POLYGCD. If you finish this and you want to try something else, write a procedure called POLYPID that takes as its input an arbitrary list of polynomials and returns a single polynomial that generates the ideal generated by all the polynomials in the list. So, for example, you should get the following relations:

```
POLYGCD(x^4-1,x^6-1) = x^{2}-1
```

and

```
POLYPID([x^3-3*x+2,x^4-1,x^{6}-1]) = x-1.
```

Note that for the last procedure it wouldn't necessarily have to return $x - 1$. Any non-zero scalar multiple of $x - 1$ would do just as well.

Section 1.5 of the textbook should have the necessary algorithms for doing both of these in pseudocode, although to do POLYPID you have to be a little comfortable with the concept of a recursive algorithm.