Mathematics 308—December 31, 1995

Introduction to PostScript

PostScript is a programming language which is **stack-based** and **interpreted**. It is primarily used for graphics and text-processing, but in fact it is a general programming language, somewhat reminiscent of the one used to program HP calculators, and can be used without trying to draw anything at all. There are several interpreters available for it. One is **dpsexec** supplied by Adobe, the company that introduced **PostScript**, that runs on several UNIX computers. But there is also one in the public domain called **ghostscript**, which is available on a variety of machines, including PC's running MSWindows. Once installed, **ghostscript** is called by typing `gs`.

One of these interpreters can be used simply as a calculator.

Calculations in **PostScript** take place according to the conventions of **reverse Polish notation** or **RPN**, which means that operators are applied to the numbers just preceding them. For example, the simple expression $3 + 4$ becomes in RPN $3\ 4\ +$. This notation could be used in ordinary mathematics, and has the advantage that it eliminates the need for parentheses. Here are some more simple examples:

Expression	RPN form
$3 + 12$	$3\ 12\ +$
$3 - 12$	$3\ 12\ -$
$3 * 12$	$3\ 12\ *$
$3/12$	$3\ 12\ /$
$(8 + 5) * 7$	$8\ 5\ +\ 7\ *$
$7 * (8 + 5)$	$7\ 8\ 5\ +\ *$
$3 + 7 * (5 + 8)$	$3\ 7\ 5\ 8\ +\ *\ +$

One curious fact that is not well appreciated is that all forms of calculation probably take place implicitly in this fashion, and that it therefore has some claim to being more natural than the algebra we use more commonly. For example, when we see a complicated expression like $(3 + 7 * (5 + 8))$ we presumably first scan it from left to right until we have seen how the grouping by parentheses takes place, and then do the calculation, here perhaps in a kind of backwards pass. In effect, we probably construct in our heads something like the RPN version of the expression before we evaluate it. At any rate, computers generally do exactly this when they evaluate arithmetic expressions. The advantage of using RPN directly is that there expressions can be evaluated on a strict left-to-right scan, with no looking ahead necessary to understand grouping—the calculations are **local**, involving only neighbouring terms. The **stack** is where the data in an expression is held until all calculations are done. It is an array whose length grows and shrinks as calculations proceed. It grows at the end called the **top**. For example, in evaluating the expression above here is a trace of how the stack looks:

3

$3\ 7$

$3\ 7\ 5$

$3\ 7\ 5\ 8$

$3\ 7\ 5\ 8\ +$

$3\ 7\ 13$

$3\ 7\ 13\ *$

$3\ 91$

$3\ 91\ +$

94

All calculations take place on the stack, and on the whole calculations affect only the numbers at the top.

A number is put on the stack by entering it. Two or more numbers must be separated by spaces (including carriage returns). An operation is performed by entering its name, and this will usually immediately replace the numbers affected by the result of the operation. At any point in a **PostScript** calculation we can see what is on the stack by entering the command `stack`, which displays everything on the stack from top to bottom, and does not change any of the items there.

For example, suppose you want to calculate the length of the vector $(9, 7)$, which is

$$\sqrt{9 * 9 + 7 * 7}\ .$$

The following is a record of a **ghostscript** session that does this, with some extra stuff showing how things are proceeding.

```
GS>9 9 mul
GS<1>stack
81
GS<1>7 7 mul
GS<2>stack
49
81
GS<2>add
GS<1>stack
130
GS<1>sqrt
GS<1>stack
11.4018
GS<1>
```

The program **ghostscript** has a prompt for input that looks like this:

```
GS<2>
```

which means that there are two items on the stack. In **PostScript**, operations + etc. are words `add` etc. In this example, we first put two copies of the number 9 on the stack and multiply them together; the copies of 9 disappear and the number 81 then sits on the stack. We then do the same for the number 7, and then both 49 and 81 are on the stack. We then add the two numbers on top of the stack by entering the command `add`, which replaces the top two numbers by their sum. By entering `sqrt` we replace this in turn by the square root of the number on the top of the stack.

It is not necessary to display results as we go. Here is a more succinct run that does the same thing.

```
GS>9 9 mul 7 7 mul add sqrt
GS<1>
```

Mathematical operators: mul, add, sub, div, sqrt, atan, cos, sin, neg.

Exercises. Calculate the length of the three-dimensional vector $(-1, 5, 7)$.

Find the polar coordinates of the two-dimensional vector $(7, -9)$.