A **Countdown** conundrum

Carol Vorderman replaced by a machine? The very idea. But here, Daniel Norris-Jones and Julian Sweeting dare to ponder, as Mike Mudge sorts the vowels from the consonants.

This month's project is proposed by Daniel Norris-Jones and Julian Sweeting of Wheldrake, Yorkshire (Dan@akqa.com).

Carol Vorderman or machine?
Those of you who have watched the TV show, Countdown, will know that Carol Vorderman is not infallible when it comes to the numbers game. Now and then she is unable to solve the problem and this raises the question: "Is it *always* possible?"

The Countdown numbers game requires the contestants to pick six cards. Each card has a number on the back. The cards are arranged into four rows. The top row contains the numbers 10, 25, 50, 75 and 100. The other three rows have cards from one to ten.

A random number generator then creates a target number (an integer between 0 and 999) and the contestants must then use the six numbers and the four operators (+, -, x, ÷) to create a number as close to the target as possible. The contestants have 30 seconds in which to achieve this target, using only pencil, paper and mental power. Carol, on the other hand, gets a little longer because the contestants prove their solutions first.

There are two ways to approach this problem: intelligence, or brute force and ignorance. The Artificial Intelligence solution may only be as good as Carol Vorderman, and until the ADI Dynamic Link Library is available it would prove difficult to implement. So this results in the solution at which computers are best. Try every possible combination of numbers and operators and then you will know whether it is possible to achieve the target.

As a schoolboy, Julian Sweeting attempted this on an Atari 8-bit home computer. Naturally he suffered from low computing power and limited knowledge. He did, however, identify the problem of parenthesis which complicated the number of permutations and combinations of operators and numbers. Initial estimates of the number of potential calculations were in the order of tens of millions, far beyond the home computing power of the eighties.

Five years on, while learning to program LISP, Sweeting came across Reverse Polish Notation and recognised that it removed the need to consider parenthesis. Obviously this was the way to tackle the problem. Some time later, during a road trip around America, he happened to discuss the problem with his fellow traveller, programmer Daniel Norris-Jones. Their appetite for solving this problem was whetted and the project sparked into life. Travelling to LA from Las Vegas, the two applied the limited processing power they had available (two Psion 3a organisers) to parts of the problem.

The essential aspects of the problem are as follows: there are six numbers and so there are 6! = 6 * 5 * 4 * 3 * 2 * 1 = 720 possible ways of ordering the numbers, i.e.

```
1-       1,2,3,10,25,50
2-       2,1,3,10,25,50
```

p258 ➢

```
3-       2,3,1,10,25,50
         ......
719-     ...
720-     50,25,10,3,2,1
```

Ignoring parenthesis, an operator may be placed between each number pair. Hence five operators, each of which can take four values, 4 to power 5 or 4 * 4 * 4 * 4 * 4 = 1,024. That is:

```
1              +++++
2              ++++-
3              ++++*
.              ....
1,023          *////
1,024          /////
```

The Psion Organiser was used to attempt simple problems not requiring parenthesis. Because limited battery life made it possible to calculate only a few tens of numbers per second, the 700,000 was out of the question.

The road trips continued and in the desert near Roswell, Sweeting and Norris-Jones had their first inclination of whether the problem could be solved within 30 seconds. They required a compiled language (100 times faster than the interpreted OPL from Psion) run on a fast computer, perhaps a few hundred times faster than a Psion. The parenthesis problem proved simple. Reverse Polish Notation showed there are only ten ways to arrange the operators and operand, which brings the total to approximately seven million numbers to calculate in 30 seconds.

The two programmers arrived in Albany, New York, where they had access to some "real" computers (IBM RISC 6000 workstations). Not all solutions to the numbers game require all six numbers to be used. It is therefore necessary to check intermediate results to determine whether you have the answer. Successive calculations differ only slightly, so only the change requires calculation. This allows intermediate results to throw a helping hand to the floundering processor. For example, given the numbers 1, 2, 3, 10, 25, 50, the calculations may be done as follows:

```
check 50 + 25 + 10 + 3 + 2 - 1
check 50 + 25 + 10 + 3 + 1 - 2
                10 calculations
```

As can be seen, the calculations are reduced if the intermediate result is stored.

```
i = 50 + 25 + 10 + 3
check i
check i + 2 - 1
check i + 1 - 2
                7 calculations
```

This may be applied simply in the nesting of the code and reduces the number of calculations per combination from five to between two and five.

At this stage, the solution was within reach. The code had been rewritten in C and was ready to go. The interpreter of the Psion had flagged overflows and these were dealt with easily. However, the Unix C compiler was not so accommodating. Overflows could go unnoticed and hence reproduce spurious results. Sweeting and Norris-Jones had come across a question which must have been asked by every serious programmer: "How do I detect integer overflow?"

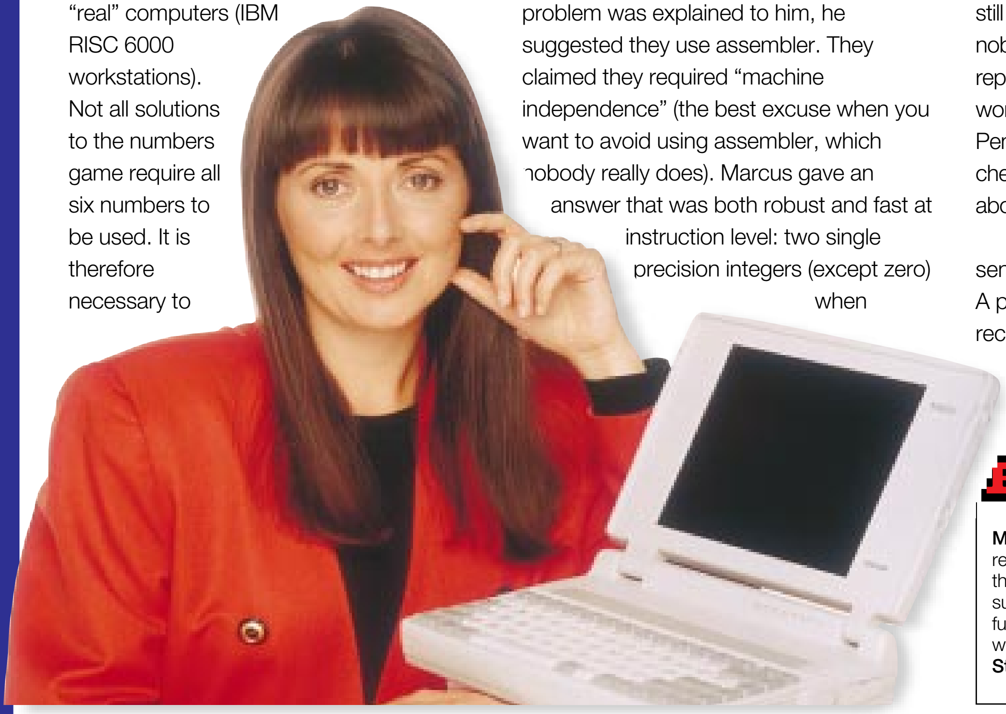
Fortunately, at 3am that night they found Marcus, a diehard programmer, in an Albany bar. When the whole Countdown problem was explained to him, he suggested they use assembler. They claimed they required "machine independence" (the best excuse when you want to avoid using assembler, which nobody really does). Marcus gave an answer that was both robust and fast at instruction level: two single precision integers (except zero) when operated on with +, -, * or / cannot be larger than a double precision integer. So use single precision integers throughout and if the answer requires any of the bits of double precision, the operation has overflowed. The code was complete.

The program executed and found solutions within five seconds. When given a problem that was impossible to calculate, the solution required inspection of all the seven million combinations. In these cases the RISC 6000 completed the job in 25 seconds: a complete success.

The performance of the program was of interest and a small routine was created to simulate the picking of numbers. A reasonable sample would be required to analyse the performance properly, but at up to 25 seconds a game, such a sample would require a few hours of runtime. Unfortunately, as these machines were for university use, a different kind of access was required. Fortunately access was available, but not in the physical sense. The code was sent to a machine back in LA using ftp, and this machine was remotely set up to execute the program for 3,678 seconds every night from midnight. Generally, Unix machines are on day and night, so exclusive processor use could almost be guaranteed at that time. The NY workstations were set up for remote viewing of the LA machine's processor occupancy. The scrolling bar chart changed from a thin line to a solid black rectangle. The processor was running flat out.

That was October 1995. The program is still running with a log file of a few megabytes: nobody likes an idle computer. We can now replace Carol Vorderman if we wish. A Unix workstation does the trick, but nowadays a Pentium should be sufficient and a pretty cheap replacement. The next task? How about trying to replace Richard Whiteley?

Investigations of this problem should be sent to Mike Mudge at the address below. A prize will be awarded to the best entry received by 1st November. (SAE for return of entries, please.)



PCW Contact

Mike Mudge welcomes correspondence from readers on any subject within the areas of number theory and computational maths, together with suggested subject areas or specific problems for future articles. Email numbers@pcw.co.uk or write to Mike at **22 Gors Fach, Pwyll-Trap, St Clears, Carmarthenshire SA33 4AQ.**