Under the counter

Mark Whitehorn shows how to cater for counters and fix up functions.

Several strings have been weaving their way through this column and last month I dealt with connections between PDAs and the internet. This month, I will take up the thread on programming.

I ended the April column on a low note: in fact, a whole series of low notes and the Psion was well on its way to playing the 'Flight of the Bumble Bee'. If you've experimented with altering the value for the Pitch% variable with a Do...Until loop to make multiple beeps, you may be wondering if there's an easy way of finding out how many beeps it made in total. Yes there is a way and it's a useful concept, too.

Add the following elements (shown in red) to the basic program:

```
PROC Penguin :
Local Pitch%, Counter%
Pitch% = 500
Counter% = 1
Do
Print Pitch%, Counter%
BEEP 1, Pitch%
Pitch% = Pitch% + 20
Counter% = Counter% + 1
      Until Pitch% > 600
Pause 50
ENDP
```

A new variable, Counter%, has been declared and set to a value of 1. When the Do...Until loop starts, the current value of Pitch% (500) and of Counter% (1) are printed to the screen. The program continues through the loop causing a 500-pitch beep, increasing the Pitch% value by 20 to 520 and the Counter% value by 1.

The loop repeats, printing the current values in the variables which are 520 and 2 respectively. When the value of Print% exceeds 600 the program stops and shown on-screen is the number of times a beep was sounded. A refinement would be to edit the Print line to:

```
Print Pitch%, "There were",↙
 Counter%, "beeps."
```

(Key: ↙ code string continues)

Counters can be seriously useful in programming, either to tell you how many times something has happened or to control how many times something *should* happen. In the latter case, the last Until line of a program might read:

```
Until Counter% = 25
```

Adding the printing feature to the program makes it run more slowly than before. Curious to know how much slower it was, I decided to run the program again without the printing.

Programming is full of situations like this. Essentially, I want the program to ignore the line of code that initiates the printing so that I can see how the program runs without it. The inefficient way to do this is to delete the line in question, which is less than ideal because you'll probably want to put it back later. The efficient way is to make it a REM statement.

A REM statement starts with the 'word' REM — any line of code starting like this is interpreted by the program as a REMark for humans to read and not to be carried out by the computer. Editing the Print line to read

```
REM Print Pitch%, "There↙
 were", Counter%, "beeps."
```

(Key: ↙ code string continues)

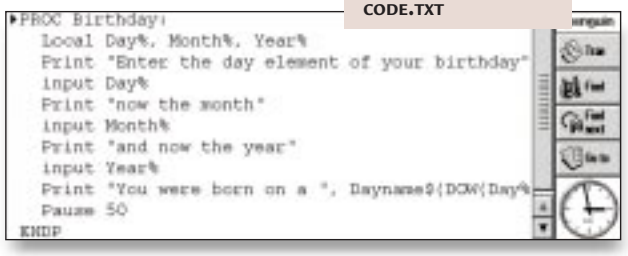
has the effect of stopping the print operation. Running the program again reveals that it runs much faster. This is not too surprising a result: printing to the screen is known to be, in many programming languages, a time-consuming operation (as are most input/output operations). To restore the printing function, simply delete REM from the start of the line.

Introducing functions

On which day of the week does your birthday fall this year?... and in the year 2000?... and in ten years time? What we need is a program to answer this important question [Fig 1], so this project introduces functions.

A function is a pre-defined piece of code which when activated during a program returns a numerical value. I'll use two of the functions built in to OPL: DOW (standing for Day Of the Week) and DayName$. DOW takes in three numbers and generates one number from those. For example, if you feed DOW with 10, 12 and 1997, it will give you 3 and if you give it 14, 2, 1999, it returns 7.

▼FIG 1 THE BIRTHDAY PROGRAM: ALL THE PROGRAM SNIPPETS ARE ON OUR COVER CD IN A FILE CALLED CODE.TXT



A simple program will test this:

```
PROC DOWtest:
      Print DOW(10,12,1997)
      Pause 50
ENDP
```

Changing the numbers will probably generate a different answer but that answer will always be between 1 and 7 and it should come as no surprise that these equate to the days of the week; so 1 maps to Monday, 2 to Tuesday etc.

These numerical values are not ideal, though. What if the users of your program think that Sunday is the first day of the week and are therefore convinced that your program didn't work? Some means of converting the number to a text string is required and for this we turn to the DayName$ function.

This one you can test with the following program:

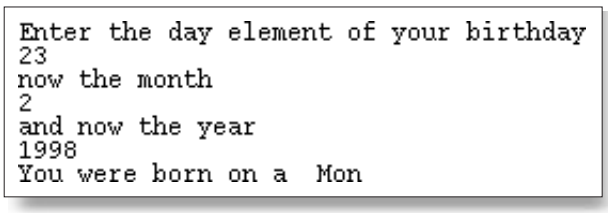
```
PROC DNtest:
      Print DayName$(3)
      Pause 50
ENDP
```

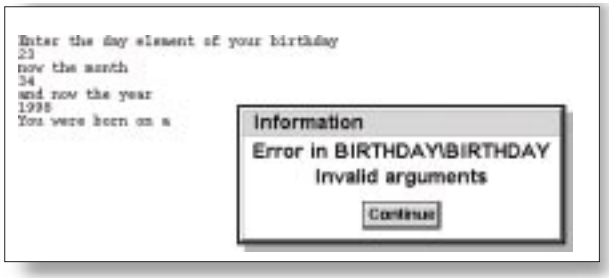
You could also use a variable as a counter *(as described above)* to run through the available options:

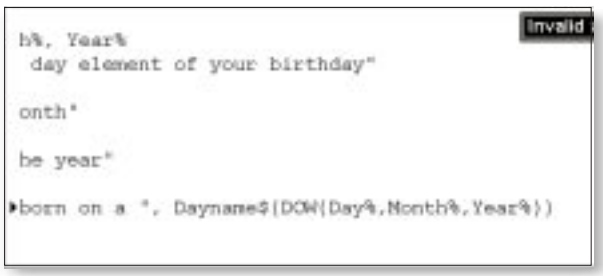
```
PROC DNTest:
Local Count%
Count% = 0
Do
      Count% = Count% + 1
      Print DayName$(Count%)
      Pause 10
Until Count% = 7
ENDP
```

GAME ON FOR THE 5

Good news for Defender fans. As you may recall from the March column, I reviewed De3ender, the excellent Defender clone game that runs on the Psion 3a, C and 3MX – just in case you missed it, it's on our cover-mounted CD again this month. I received an email from De3ender's author, Mark Wheadon, who says he's working on a version for the Series 5. When he mailed me, he'd just run some tests to check the frame rate, amongst other things, and told me the results 'are very encouraging and the Series 5 version is going to be seriously smooth and responsive'. Mark says he'll let me know as soon as it's ready to roll but the impatient should keep an eye on his web site at i.am/MarkSoft. Mark is also responsible for two games for the Siena: Siender (the Defender clone) and Patsiena (Patience).







◂ **FIG 2 THE PROGRAM RUNS!**

▾ **FIG 3 (MIDDLE) THE PROGRAM BOMBS...**

▾ **FIG 4 (BOTTOM) ...AND OPL TRIES TO HELP**

which should list all the days of the week.

☛ **A program combining** these two functions, where DOW generates the number and DayName$ interprets it, can be written as follows:

```
PROC Combine:
      Print DayName$(DOW(10↙
,12,1997))
      Pause 50
ENDP
```

(Key: ↙ code string continues)

The concept of nesting functions is also introduced by this rather terse program. The function DOW is given three figures and from those it calculates the day number, which is 3. That number 3 could be placed into a variable for later use but instead I've put the DOW function inside brackets that follow the DayName$ function.

DayName$ expects to receive a number and this is what it gets from the DOW function. In other words, DOW works with the figures it's given to produce a number and that number is then used by the preceding DayName$ function. This short and sweet solution forms the core of the birthday program.

☛ **Until now,** the example programs have been for use by you, the programmer and whenever you've wanted to try something different, you have edited the code and run it again. The birthday worker-outer, however, is different in that you may wish to give it a user interface so others can use it. In order to work, the program must be able to take input from the user and incorporate it. The OPL command for this is INPUT. This halts the program, waits for typed input from the user, puts that input into a variable and then sets the program running again.

Users will need to enter a day, a month and a year, so three variables are needed for these pieces of information: in this example they are Day%, Month% and Year%. INPUT is called on three times to take input and continue, as shown below:

```
PROC Birthday:
      LOCAL Day%,Month%,Year%
   Print "Enter the day ↙
element of your birthday"
       Input Day%
       Print "now the month",
       Input Month%
  Print "and now the year",
       Input Year%
 Print "You were born on a ↙
", DayName$(DOW(Day%,↙
 Month%,Year%))
Pause 50
ENDP
```

(Key: ↙ code string continues)

☛ **In your hands,** the program works beautifully but out there in the wild it has to cope with the vagaries of users [Figs 2, 3 & 4]. Users make 'mistakes' that cause the program to lurch to an untidy halt. They type in 'Tue' for the day, 14 for the month or 99 for the year when what your program expects is the day of the month as a number, a month between one and 12 and a four-digit year. (To be pedantic, OPL on the Psion 5 can cope with fewer

READ AND WRITE

Which PDA?

Reader Neil Shaw <neil@8adel.freeserve .co.uk> asked me to imagine having £500 to spend on a palm top computer and wonders which of the following options I'd go for and why:

'I'm considering either the HP 620LX at £424 (ex VAT) or the Sharp Mobilon HC 4600 at £439 (ex VAT). Both have good resolution colour screens but equally bad battery life. The Sharp has a built-in modem and is slightly smaller and lighter. Are there any others that you know of, either worth a look now, or soon to be introduced?'

If I had to choose between these two, I would go for the Sharp, on the grounds of its modem and its weight, but it is a matter of personal preference, depending on your own likes and dislikes.

My current two favourite machines are the LG Phenom Express and the Psion 3MX. The former runs WinCE, has a great touch-sensitive colour screen and an excellent keyboard that's 80 percent full size. On the down side is its battery life — a bit grim at only 5-6 hours — and its size (that 80 percent keyboard has to go somewhere).

My other rave, the Psion 3MX, is an elegant little machine with a small but usable keyboard and wonderful battery life. But it's monochrome, not touch sensitive and does not run WinCE. Whether or not you count these 'features' as advantages or disadvantages is up to you and the final decision almost always contains an element of compromise. At present, I'm using both these machines and love them.

I love my PDA

On the subject of PDA love, check out the letter from reader W. O. Adepoju <wadepoju@oauife.edu.ng> on our cover CD because it is too long to include in this column. Here is a man who loves his PDA and I know how he feels. I once dropped my Psion onto the stylishly hard and shiny marble floor of an airport: it was the demise of my 'travelling companion' and I was bereft.

Mr Adepoju's problem is a hardware fault that's causing his beloved Psion 3a to voraciously chew its way through backup batteries. Having spoken to Psion, there is no recourse other than to return the machine to the company for fixing/replacement, but there seems little doubt that the relationship will survive the temporary separation.

Where's the wallpaper?

The Biddle family <Biddle@wr116ye. freeserve .co.uk> or presumably one member thereof, wants to customise the Psion 5 by adding wallpaper. In the January column I mentioned TascalWPC, a Windows CE utility for changing wallpaper. It is available from the freeware site www2r.biglobe.ne.jp/~tascal/english/intex.htm but as yet I have seen nothing that does the same for the Psion 5.

• *Does anybody out there know differently?*

Advanced programming

Reader Al Richey <support@rmrsoft.com> wrote: 'I liked the intro to programming and thought I should mention that if you want to move onto more advanced stuff, Steve Litchfield has presented a programming tutorial on his 3-Lib Page.'

Steve Litchfield's name is inseparable from the word 'Psion' and his web site is always worth a browse. It is a great source of Psion-type information on all sorts of topics from mapping software to, as Al Richley says, OPL programming.

Components update

IN THE MARCH ISSUE COLUMN, I REFERRED TO THE ABOVE SITE WITH DETAILS OF MAKING CHEAP(ISH) PSION CABLES AND SOURCING COMPONENTS. UNFORTUNATELY THERE WAS A MISPRINT IN THE URL. THE CORRECT VERSION IS WWW-SP.PHY.CAM.AC.UK/~JRB25/PSICABLE.HTML

◂ FIG 5 ON THE PSION 5, YOU SHOULD FIND ICONS FOR THE 'COMPILED' VERSION OF YOUR PROGRAM WHICH, IF CLICKED, WILL RUN YOUR PROGRAM

than four digits for a year but OPL on the Series 3 cannot). It's not really the users' fault because nobody told them the rules. But from the users' point of view, the program fails to work and they aren't impressed.

There are three solutions. Firstly, you can write programs exclusively for your own use (boring). Secondly, you can berate users loudly every time they make a mistake (dangerous). Thirdly, you can add 'error trapping' to the program (gratifying). Error trapping means catching any error the user makes before that error causes the program to fail and it is this area which I will look at next time. Error trapping is by far the most satisfying solution but can lead to misplaced smugness when you think you've second-guessed all possible errors. Be warned — users are surprisingly inventive!

PCW CONTACTS

Mark Whitehorn welcomes your feedback on the PDAs column. He can be contacted via the PCW editorial office (address, page 14) or email pda@pcw.co.uk