

Stardodger II – the BCPL version

Stewart Russell shows you how to program the same game three times, in three very different languages

ONCE the Basic version of Stardodger worked to my satisfaction – which took longer than expected – the program was rewritten using Arnor's BCPL compiler. BCPL was the forerunner of the oh-so-trendy C language beloved by computer scientists and other deviants. Unlike C, BCPL is quite readable, yet it still enforces a carefully structured programming style.

This is due to its syntax and the lack of error checking. Care must be taken or the compiler will merrily churn out guff without a single beep of displeasure.

The BCPL Stardodger took far less time to write than the Basic version, mainly because all the program logic had already been worked out.

Dynamic elegance

A particularly neat feature of BCPL is the case structure – SWITCHON..INTO..CASE, used here in the collision detection routine – which is similar to, but more elegant than, Basic's ON..GOTO.

Nearly all the variables used in this program

```
Arnor BCPL compiler
Output file name? STARBCPL
-> OPTION S-B-
-> GET"ALIBHDR"
-> GET"ALIBHDR1"
-> GET"AMSDOS"
-> GET"STARDOJ.B"
->
```

```
Phase 1 complete. Tree size 15652
Phase 1 errors: 0
Phase 2 complete. Code size 9631
Phase 2 errors: 0
Code origin 370
```

Compiling the BCPL version – the dialogue

Routine	Basic lines	BCPL procs
Initialisation	20-70	start
Print title screen	90-170	start
Draw game screen	180-450	drawscr
Main game logic	470-530	start
Print game over screen	550-600	start
Print success screen	620-680	start
Wait for keypress routine	700-760	waitkey

The main routines – a comparison

are static variables; this means they are always available to any part of the program. Dynamic variables – such as *t* used in the pause procedure – disappear after being finished with. Unlike Basic, all BCPL variables and constants, known as *manifests*, have to be defined before use.

Also unlike Basic, which has string, integer and real variables, BCPL has only one type of variable – the "word", or 16 bits. This makes it ideal for implementation on a home micro.

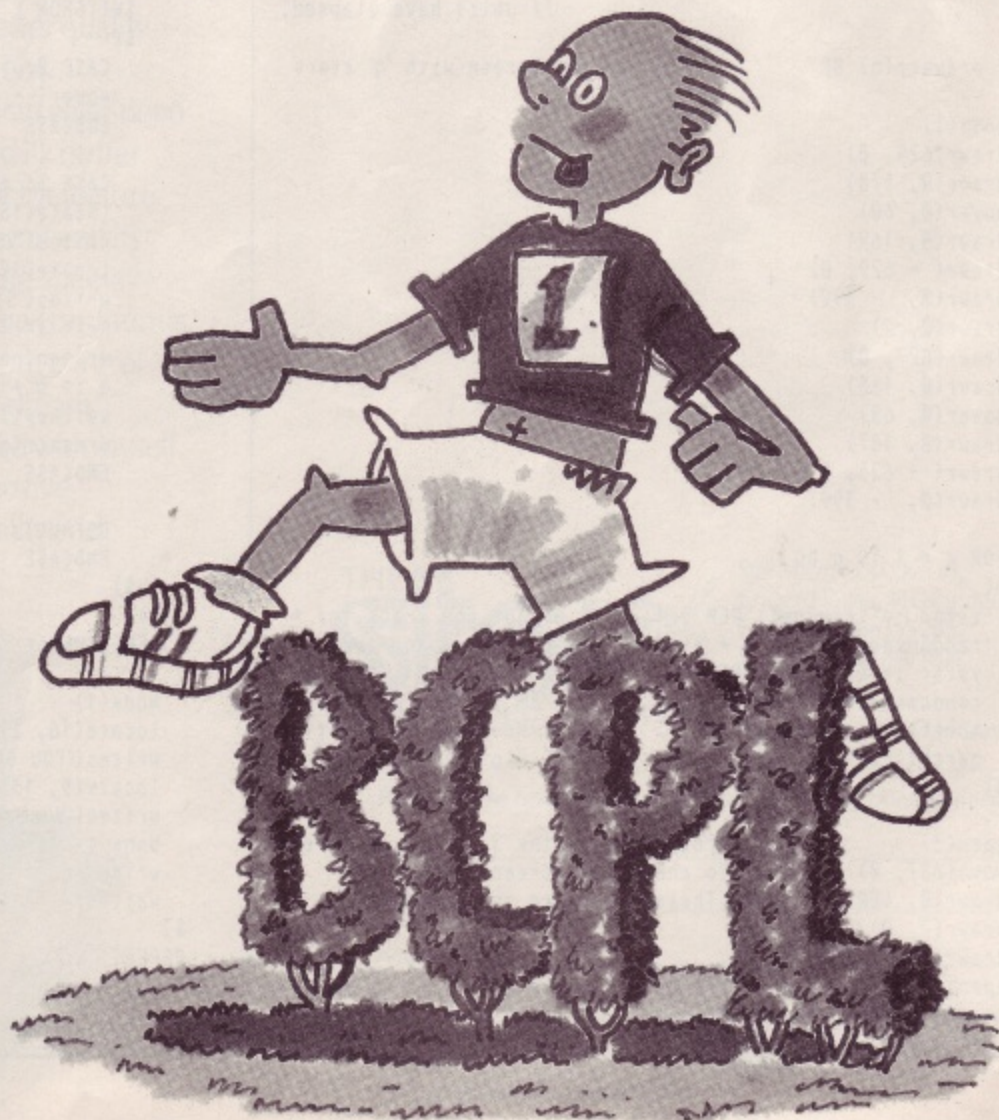
It does have some odd conventions though. For instance, the asterisk is thought of as a control character. It cannot be represented as simply * but has to be written as ** before it is accepted. Gripes aside, BCPL is a lovely language to use.

You can use any Ascii text editor for producing the source code. Indenting the text is not neces-

ary, but helps to show the levels of the program. After saving the text – call it STARDOJ.B – it may be an idea to dry run it through the compiler without GETting any of the libraries. As long as only *Undefined identifier* errors are produced the text should be OK. But beware of spelling mistakes in procedure names, as these cannot be checked for until the final compilation stage.

You must first invoke the compiler from disc, using RUN"DISC and then IBCPL. Follow the compiler dialogue in the panel, but note that minor differences may occur in the numeric values produced.

● Next month, in the final part of this series, we'll look at the assembly language version.




```
// Stardodger using Arnor's BCPL compiler.
// Written by Stewart C Russell of Edible Computers.
// Requires Alibhdr, Alibhdr1 and Amsdos libraries.

MANIFEST
$(
  star = '**'
  delay = 3          // Loop delay in 1/300ths of a sec.
$)

STATIC
$(
  increment = 5      // Number of stars added per screen
  xstar = 0          // X-position of star
  ystar = 0          // Y-position of star
  dy = 4             // Y-position increment
  q = 5              // Start no of stars per screen
  done = 0           // Number of screens completed
  next = 0           // Next screen number
  status = 0         // Status, 1 = dead, 0 = not dead
  ks = 0             // Shift key status
  c = 0              // Ink status for collision
  y = 0              // Collision detection y-pos increment
$)

LET waitkey() BE      // Prints message and waits for key
$(
  locate(8, 25)
  writes("Press any key to continue.")
  WHILE keyvalid() DO LOOP // Clear buffer
  UNTIL keyvalid() DO LOOP // Continue on keypress
$)

LET pause(length) BE  // Pauses for length/300 seconds
$(
  LET t = time()      // Get current time
  UNTIL time() EQ t + length DO LOOP // Wait until "length"
  $)                  // units have elapsed.

LET drawscr(q) BE     // Draw the screen with "q" stars
$(
  mode(1)
  drawr(629, 0)
  drawr(0, 170)
  mover(0, 60)
  drawr(0, 169)
  drawr(-629, 0)
  drawr(0, -399)
  drawr(0, 2)
  drawr(627, 0)
  drawr(0, 168)
  mover(0, 60)
  drawr(0, 167)
  drawr(-625, 0)
  drawr(0, -399)

  FOR s = 1 TO q DO
  $(
    xstar := (random() REM 561) + 50 // Get rnd x-pos for *
    randomseed := xstar + time()    // Feed random seed
    ystar := (random() REM 361) + 20 // Y-pos
    randomseed := randomseed - (xstar REM ystar + q) // Seed
    move(xstar, ystar)              // Move to rnd position
    gwchr(star)                     // Plot a * there
  $)

  gpen(3)                          // Draw lines in ink 3 at end of screen
  move(637, 0)                      // to check for screen completion
  drawr(0, 400)                    // (These lines are invisible)
  drawr(2, 0)
  drawr(0, -400)
  gpen(1)                          // Set pen to white again
  move(0, 200)                     // Move to line start position
$)

```

```
$)

LET start() BE        // *** Main Routine ***
$(
  $(
    mode(1)
    border(0, 0)
    ink(0, 0, 0)
    ink(1, 26, 26)
    ink(3, 0, 0)      // Set up inks and mode
    locate(16, 1)
    writes("Stardodger") // Print title screen
    locate(1, 5)
    writes("Avoid the killer Asterisks, and seek the")
    locate(8, 6)
    writes("wondrous Nextscreen Gap !")
    locate(12, 13)
    writes("Use SHIFT to climb")
    pen(2)
    locate(3, 18)
    writes("Written in BCPL by Stewart C Russell")
    locate(9, 19)
    writes("Edible Computers 23/4/88")
    pen(1)
    waitkey()          // Press any key message
    status := 0        // Reset pointers
    q := 5             // to screen 0, status = alive
    drawscr(q)         // Draw screen 1 (five stars)
  $(
    ks := 0            // Clear key status variable
    drawr(4, dy)       // Draw line unit
    pause(delay)       // To allow for reactions
    ks := inkey(21)     // Get shift key status
    TEST ks EQ -1 THEN dy := 4 ELSE dy := -4 // Move up
    y := dy / 2        // Get y-pos in front of line
    c := gtestr(2, y)  // Test point in front of line

    SWITCHON c INTO    // Act on ink no. accordingly
    $(
      CASE 0: y := -1 * y // If ink 0
      mover(-2, y)       // go back to old coords.
      ENDCASE

      CASE 3: mode(1)    // If ink 3
      locate(16, 1)     // congratulate player
      writes("WELL DONE") // on completion.
      locate(10, 13)
      writes("Stand by for Screen ")
      next := (q / increment) + 1
      writen(next)      // Print next screen no
      q := q + increment // Increase no of stars
      waitkey()
      drawscr(q)        // Draw the next screen
      ENDCASE

      DEFAULT: status := 1 // Default to dying
      ENDCASE
    $)
  $)
  REPEATUNTIL status NE 0 // Repeat loop while not dead

  mode(1) // Player is dead if we've got to here
  locate(16, 1)
  writes("YOU GOOFED")
  locate(5, 13)
  writes("Number of Screens completed = ")
  done := (q / increment) - 1
  writen(done) // Print no of screens completed
  waitkey()

  $)
  REPEAT // Repeat outer loop of "start"
  $)

```