OPL

EXAMPLE PROGRAMS

© Copyright Psion Computers PLC 1997

This manual is the copyrighted work of Psion Computers PLC, London, England.

The information in this document is subject to change without notice.

Psion and the Psion logo are registered trademarks. Psion Series 5, Psion Series 3c, Psion Series 3a, Psion Series 3 and Psion Siena are trademarks of Psion Computers PLC.

EPOC32 and the EPOC32 logo are registered trademarks of Psion Software PLC.

© Copyright Psion Software PLC 1997

All trademarks are acknowledged.

CONTENTS

OPL

This document contains example programs written in OPL. The programs are not intended to demonstrate all the features of OPL, but they should give you a few hints. To find out more about a particular command or function, refer to the ‘Alphabetic Listing’ section of the ‘Glossary.pdf’ document.

There are some further example programs in the ‘Advanced Topics’ section of the ‘Advanced.pdf’ document.

WHEN YOU’RE TYPING IN

- You can type procedures in all uppercase, all lowercase or any mixture of the two. Be careful with character codes, though - %A is different to %a.
- When there is more than one command or function on a line, separate each one with a space and colon - for example:
  `CLS :PRINT “hello” :GET`
  However, the colon is optional before a `REM` statement for example:
  `CLS REM Clears the screen`
  and
  `CLS :REM Clears the screen`
  are both OK.
- Put a space between a command and the arguments which follow it - for example `PRINT a$`. But don’t put a space between a function and the arguments in brackets - for example `CHR$(16)`.
- It doesn’t matter how many spaces or tabs you have at the beginning of a line.

ERRORS

The following programs do not include full error handling code. This means that they are shorter and easier to understand, but may fail if, for example, you enter the wrong type of input to a variable.

If you want to develop other programs from these example programs, it is recommended that you add some error handling code to them. See the ‘Error Handling’ section of the ‘Advanced.pdf’ document for further details.

COUNTDOWN TIMER

❺ For the Series 5:

```
PROC timer:
   LOCAL min&,sec&,secs&,i%
   sec&=1
   dINIT “Countdown timer”
   dLONG min&,“Minutes”,0,59
   dLONG sec&,“Seconds”,0,59
   dBUTTONS “Cancel”,-27,“Start”,%s
   IF DIALOG=%s
      FONT 12,16
      secs&=sec&+60*min&
      WHILE secs&
            PAUSE -20                    REM a key gets us out
            IF KEY
                  RETURN
            ENDIF
            secs&=secs&-1
            AT 20,6 :PRINT NUM$(secs&/60,-2);“m”
            AT 24,6 :PRINT NUM$(mod&:(secs&,int(60)),-2);“s”
      ENDWH
      DO
            BEEP 5,300
            PAUSE 10
            IF KEY :BREAK :ENDIF
            i%=i%+1
      UNTIL i%=10
   ENDIF
ENDP

PROC mod&:(a&,b&)
   REM modulo function
   REM computes (a&)mod(b&)
   RETURN a&-(a&/b&)*b&
ENDP
```

❸ For the Series 3c and Siena:

```
PROC timer:
   LOCAL min&,sec&,secs&,i%
   CACHE 2000,2000
   sec&=1
   dINIT “Countdown timer”
   dLONG min&,”Minutes”,0,59
   dLONG sec&,”Seconds”,0,59
   dBUTTONS “Cancel”,-27,”Start”,13
   IF DIALOG=13
      STATUSWIN ON
      FONT 11,16
      secs&=sec&+60*min&
      WHILE secs&
            PAUSE -20                     REM a key gets us out
            IF KEY
                  RETURN
            ENDIF
            secs&=secs&-1
            AT 20,6 :PRINT NUM$(secs&/60,-2);”m”
            AT 24,6 :PRINT NUM$(mod&:(secs&,int(60)),-2);”s”
      ENDWH
      DO
            BEEP 5,300
            PAUSE 10
            IF KEY :BREAK :ENDIF
            i%=i%+1
      UNTIL i%=10
   ENDIF
ENDP
```

```
PROC mod&:(a&,b&)
   REM modulo function
   REM computes (a&)mod(b&)
   RETURN a&-(a&/b&)*b&
ENDP
```

DICE

When the program is run, a message is displayed saying that the dice is rolling. You then press a key to stop it. A random number from one to six is displayed and you choose whether to roll again or not.

```
PROC dice:
   LOCAL dice%
   DO
      CLS :PRINT “DICE ROLLING:”
      AT 1,3 :PRINT “Press a key to stop”
      DO
            dice%=(RND*6+1)
            AT 1,2 :PRINT dice%
      UNTIL KEY
      BEEP 5,300
      dINIT “Roll again?”
      dBUTTONS “No”,%N,“Yes”,%Y
   UNTIL DIALOG<>%y
ENDP
```

RANDOM NUMBERS

In this example, the RND function returns a random floating-point number, between 0 and 0.9999999... It is then multiplied by 6, and 1 is added, to give a number from 1 to 6.9999999... This is rounded down to a whole number (from 1 to 6) by assigning to the integer `dice%`.

BIRTHDAYS

This procedure finds out on which day of the week people were born.

```
PROC Birthday:
   LOCAL day&,month&,year&,DayInWk%
   DO
      dINIT
      dTEXT “”,“Enter your date of birth”,2
      dTEXT “”,“Use numbers, eg 23 12 1963”,$202
      dLONG day&,“Day”,1,31
      dLONG month&,“Month”,1,12
      dLONG year&,“Year”,1900,2155
      IF DIALOG=0
            BREAK
      ENDIF
      DayInWk%=DOW(day&,month&,year&)
      CLS :PRINT DAYNAME$(DayInWk%),day&,month&,year&
```

```
    dINIT “Again?”
    dBUTTONS “No”,%N,“Yes”,%Y
  UNTIL DIALOG<>%y
ENDP
```

The DOW function works out what day of the week, from 1 to 7, a date is. The DAYNAME$ function then converts this to MON, TUE and so on. MON is 1 and SUN is 7.

DATA FILES

The following module works on a data file called DATA, containing names, addresses, post codes and telephone numbers. It assumes this file has already been created with a statement like this:

```
CREATE “DATA”,A,nm$,ad1$,ad2$,ad3$,ad4$,tel$
```

❺ To use a database created with the Data application, see the ‘Series 5 Database Handling’ section of the ‘Database.pdf’ document.

❸ To use the DATA file which the Database application uses, you need to open “\DAT\DATA.DBF”.

The first procedure is the controlling, calling procedure, offering you choices. The next two let you add or edit records.

```
PROC files:
  GLOBAL nm$(255),ad1$(255),ad2$(255)
  GLOBAL ad3$(255),ad4$(255),tel$(255),title$(30)
  LOCAL g%
  OPEN “DATA”,A,nm$,ad1$,ad2$,ad3$,ad4$,tel$
  DO
    CLS
    dINIT “Select action”
    REM !!Swap prompt and body in dTEXT for Series 3c and Siena!!
    dTEXT “Add new record”,“”,$402
    dTEXT “Find and edit a record”,“”,$402
    g%=DIALOG
    IF g%=2
        add:
    ELSEIF g%=3
        edit:
    ENDIF
  UNTIL g%=0
  CLOSE
ENDP

PROC add:
  nm$=“” :ad1$=“” :ad2$=“”
  ad3$=“” :ad4$=“” :tel$=“”
  title$=“Enter a new record”
  IF showd%:
    APPEND
  ENDIF
ENDP
```

```
PROC edit:
  LOCAL search$(30),p%
  dINIT “Find and edit a record”
  dEDIT search$,“Search string”,15
  IF DIALOG
    FIRST
    IF FIND(“*”+search$+“*”)=0
        ALERT(“No matching records”)
        RETURN
    ENDIF
    DO
        nm$=A.nm$ :ad1$=A.ad1$ :ad2$=A.ad2$
        ad3$=A.ad3$ :ad4$=A.ad4$ :tel$=A.tel$
        title$=“Edit matching record”
        IF showd%:
            UPDATE :BREAK
        ELSE
            NEXT
        ENDIF
        FIND(“*”+search$+“*”)
        IF EOF
            ALERT(“No more matching records”)
            BREAK
        ENDIF
    UNTIL 0
  ENDIF
ENDP

PROC showd%:
  LOCAL ret%
  dINIT title$
  dEDIT nm$,“Name”,25
  dEDIT ad1$,“Street”,25
  dEDIT ad2$,“Town”,25
  dEDIT ad3$,“County”,25
  dEDIT ad4$,“Postcode”,25
  dEDIT tel$,“Phone”,25
  ret%=DIALOG
  IF ret%
    A.nm$=nm$ :A.ad1$=ad1$ :A.ad2$=ad2$
    A.ad3$=ad3$ :A.ad4$=ad4$ :A.tel$=tel$
  ENDIF
  RETURN ret%
ENDP
```

RE-ORDER

When you use the Data application and enter or change an entry, it goes to the end of the database file. However, **if**, in your address book, each entry begins with a person's second name - for example, `Tate, Hazel` - you can use this program to re-order all of the entries. This doesn't change the way you find an entry, but after running it you can step through it like a paper address book, or print it out neatly ordered.

This procedure can be used as required for any data file in internal memory or on memory disk for the Series 5 or on Ram SSDs for the Series 3c. For the Series 3c, note that if used on a data file held on a Flash SSD it would use up disk space each time you run it. The dialog it shows is set to show data files used by Data.

You can adapt this procedure to sort other types of data files in other ways.

Note that on the Series 5, this would be better done with the more advanced features available in the Database OPX. See the 'Using OPXs on the Series 5' section of the 'Advanced.pdf' document for more details of this. You could also use restriction of files by UID in the dFILE keyword to restrict to databases only.

```
PROC reorder:
  LOCAL last%,e$(255),e%,lpos%,n$(255),c%
  n$="\dat\*.dbf"
  dINIT "Re-order Data file"
  dFILE n$,"Filename",0
  IF DIALOG                     REM returns 0 if cancelled
    OPEN n$,a,a$
    LAST :last%=POS
    IF COUNT>0
        WHILE last%<>0
            POSITION last% :e%=POS
            e$=UPPER$(a.a$)
            DO
                IF UPPER$(a.a$)<e$
                    e$=UPPER$(a.a$) :e%=POS
                ENDIF
                lpos%=POS :BACK
            UNTIL pos=1 and lpos%=1
            POSITION e%
            PRINT e$
            UPDATE :last%=last%-1
        ENDWH
    ENDIF
    CLOSE
  ENDIF
  GET
ENDP
```

If you try to reorder a file which is already open (i.e. shown in bold on the System screen) you will see a "*File*' is in use' ('File or device in use' on the Series 3c) error. You should close the file and then try again.

STOPWATCH

Here is a simple stopwatch with lap times. Note that the Psion switches off automatically after a time if no keys are pressed; you may want to disable this feature (from the Control Panel in the System screen on the Series 5 or with the ‘Auto switch off’ option in the System screen on the Series 3c) before running this program.

Each timing is only accurate to within one second, as the procedure is based on the SECOND function.

```
PROC watch:
  LOCAL k%,s%,se%,mi%
  FONT 11,16
  AT 20,1 :PRINT “Stopwatch”
  AT 15,11 :PRINT “Press a key to start”
  GET
  DO
    CLS :mi%=0 :se%=0 :s%=SECOND
    AT 15,11 :PRINT “  S=Stop, L=Lap  ”
loop::
  k%=KEY AND $ffdf               REM ensures upper case
  IF k%=%S
    GOTO pause::
  ENDIF
  IF k%=%L
    AT 20,6 :PRINT “Lap: ”;mi%;“:”;
    IF se%<10 :PRINT “0”; :ENDIF
    PRINT se%;“ ”;
  ENDIF
  IF SECOND<>s%
    s%=SECOND :se%=se%+1
    IF se%=60 :se%=0 :mi%=mi%+1 :ENDIF
    AT 17,8
    PRINT “Mins”,mi%,“Secs”,
    IF se%<10 :PRINT “0”; :ENDIF
    PRINT se%;“ ”;
  ENDIF
  GOTO loop::
pause::
  mINIT
  mCARD “Watch”,“Restart”,%r,“Zero”,%z,“Exit”,%x
  k%=MENU
  IF k%=%r
    GOTO loop::
  ENDIF
  UNTIL k%<>%z
ENDP
```

INSERTING A NEW LINE IN A DATABASE

If you insert a new label in a database, the entries will no longer match up with the labels. Rather than using the ‘Update’ option on every entry, to insert a suitable blank line in each one, you can use this program to do this for the entire data file.

The Data application allows you to use as many lines (fields) as you want in an entry (record); OPL can only access 32 fields. This program only lets you insert a new field in the first 16 fields, although you can adapt the program simply to check up to 31 fields.

If, in Data, you enter a line longer than 255 characters, it is stored as two fields, with a character of code 20 at the start of the second field. This program correctly handles any such fields.

The program checks that the 17th field is blank, as it will be overwritten by what was the 16th field. If a long entry has a 17th field, and it contains text, the program skips this entry. The rest of longer entries - even if there are more than 32 fields will be unchanged.

If you insert a new field at a position below the last label, Data will not show it, even when using ‘Update’.

The maximum record length in OPL is 1022 characters. The OPEN command will display a ‘Record too large’ error if the file contains a record longer than this.

```
PROC label:
   LOCAL a%,b%,c%,d%,s$(128),s&,i$(17,255)
   s$=“\dat\*.dbf”
   dINIT “Insert new field”
   dFILE s$,“Data file”,0
   dLONG s&,“Break at line (1-16)”,1,16
   IF DIALOG
      OPEN s$,A,a$,b$,c$,d$,e$,f$,g$,h$,i$,j$,k$,l$,m$,n$,o$,p$,q$
      c%=COUNT :a%=1
      WHILE a%<=c%
            AT 1,1 :PRINT “Entry”,a%,”of”,c%,
            IF A.q$=“”                    REM Entry (hopefully) not too long
                  i$(1)=A.a$ :i$(2)=A.b$ :i$(3)=A.c$ :i$(4)=A.d$
                  i$(5)=A.e$ :i$(6)=A.f$ :i$(7)=A.g$ :i$(8)=A.h$
                  i$(9)=A.i$ :i$(10)=A.j$ :i$(11)=A.k$ :i$(12)=A.l$
                  i$(13)=A.m$ :i$(14)=A.n$ :i$(15)=A.o$ :i$(16)=A.p$
                  d%=0 :b%=0
                  WHILE d%<s&+b%          REM find field to break at
                        d%=d%+1
                        IF LEFT$(i$(d%),1)=CHR$(20)      REM line>255...
                              b%=b%+1     REM ...so it’s 2 fields
                        ENDIF
                  ENDWH
                  b%=17
                  WHILE b%>d%             REM copy the fields down
                        i$(b%)=i$(b%-1) :b%=b%-1
                  ENDWH
                  i$(d%)=“”               REM and make an empty field
                  A.a$=i$(1) :A.b$=i$(2) :A.c$=i$(3) :A.d$=i$(4)
                  A.e$=i$(5) :A.f$=i$(6) :A.g$=i$(7) :A.h$=i$(8)
                  A.i$=i$(9) :A.j$=i$(10) :A.k$=i$(11) :A.l$=i$(12)
                  A.m$=i$(13) :A.n$=i$(14) :A.o$=i$(15) :A.p$=i$(16)
            A.q$=i$(17)
        ELSE
            PRINT “has too many fields”
            PRINT “Press a key...” :GET
        ENDIF
        UPDATE :FIRST
        a%=a%+1
    ENDWH :CLOSE
  ENDIF
ENDP
```

BOUNCING BALL

```
PROC bounce:
  LOCAL posX%,posY%,changeX%,changeY%,k%
  LOCAL scrx%,scry%,info%(10)
  SCREENINFO info%()
  scrx%=info%(3) :scry%=info%(4)
  posX%=1 :posY%=1
  changeX%=1 :changeY%=1
  DO
    posX%=posX%+changeX%
    posY%=posY%+changeY%
    IF posX%=1 OR posX%=scrx%
        changeX%=-changeX%
        REM at edge ball changes direction
        BEEP 2,600                  REM low beep
    ENDIF
    IF posY%=1 or posY%=scry%       REM same for y
        changeY%=-changeY%
        BEEP 2,200                  REM high beep
    ENDIF
    AT posX%,posY% :PRINT “0”;
    PAUSE 2                         REM Try changing this
    AT posX%,posY% :PRINT “ ”;      REM removes old ‘0’ character
    k%=KEY
  UNTIL k%
ENDP
```

CIRCLES

❺ Here is an example program for drawing circles or ellipses, filled or unfilled for the Series 5:

```
PROC draw:
  LOCAL d%
  DO
    dINIT “Draw a circle or an ellipse?”
    dBUTTONS “Circle”,%c OR $200,“Ellipse”,%e OR $200,“Cancel”,-27
    d%=DIALOG
    IF d%=%c
        circle:
    ELSEIF d%=%e
```

```
         ellipse:
      ENDIF
   UNTIL d%=0
ENDP

PROC circle:
   LOCAL x&,y&,r&,f%
   dINIT “Drawing parameters”
   x&=320 :dLONG x&,”Centre x position”,0,639
   y&=120 :dLONG y&,”Centre y position”,0,249
   r&=20 :dLONG r&,”Radius”,1,320
   f%=0 :dCHECKBOX f%,”Filled”
   dBUTTONS “Draw”,%d,“Cancel”,-27
   IF DIALOG
      gAT x&,y&
      gCIRCLE r&,f%
      GET
      gCLS
   ENDIF
ENDP

PROC ellipse:
   LOCAL x&,y&,hr&,vr&,f%
   dINIT “Drawing parameters”
   x&=320 :dLONG x&,”Centre x position”,0,639
   y&=120 :dLONG y&,”Centre y position”,0,249
   hr&=20 :dLONG hr&,”Horizontal Radius”,1,320
   vr&=20 :dLONG vr&,”Vertical Radius”,1,320
   f%=0 :dCHECKBOX f%,”Filled”
   dBUTTONS “Draw”,%d,“Cancel”,-27
   IF DIALOG
      gAT x&,y&
      gELLIPSE hr&,vr&,f%
      GET
      gCLS
   ENDIF
ENDP
```

❸ Here are **two** example programs for drawing circles - the first hollow, the second filled for the Series 3c and Siena:

```
PROC circle:
   LOCAL a%(963),c&,d%,x&,y&,r&,h,y%,y1%,c2%
   dINIT “Draw a circle”
   x&=240 :dLONG x&,“Centre x pos”,0,479
   y&=80 :dLONG y&,“Centre y pos”,0,159
   r&=20 :dLONG r&,“Radius”,1,120
   h=1 :dFLOAT h,“Relative height”,0,999
   IF DIALOG
      a%(1)=x&+r& :a%(2)=y& :a%(3)=4*r&
      c&=1 :d%=2*r& :y1%=0
```

```
     WHILE c&<=d%
          c2%=c&*2 :y%=-SQR(r&*c2%-c&**2)*h
          a%(2+c2%)=-2 :a%(3+c2%)=y%-y1%
          y1%=y% :c&=c&+1
     ENDWH
     c&=1
     WHILE c&<=d%
          c2%=c&*2 :y%=SQR(r&*c2%-c&**2)*h
          a%(2+a%(3)+c2%)=2 :a%(3+a%(3)+c2%)=y%-y1%
          y1%=y% :c&=c&+1
     ENDWH
     gPOLY a%()
   ENDIF
ENDP

PROC circlef:
   LOCAL c&,d%,x&,y&,r&,h,y%
   dINIT “Draw a filled circle”
   x&=240 :dLONG x&,“Centre x pos”,0,479
   y&=80 :dLONG y&,“Centre y pos”,0,159
   r&=20 :dLONG r&,“Radius”,1,120
   h=1 :dFLOAT h,“Relative height”,0,999
   IF DIALOG
     c&=1 :d%=2*r& :gAT x&-r&,y& :gLINEBY 0,0
     WHILE c&<=d%
          y%=-SQR(r&*c&*2-c&**2)*h
          gAT x&-r&+c&,y&-y% :gLINEBY 0,2*y%
          c&=c&+1
     ENDWH
   ENDIF
ENDP
```

If you use gUPDATE OFF after the IF DIALOG line, and gUPDATE ON before the ENDIF, the procedure will run a little faster. However, all but the smaller circles will be drawn rather jerkily, piece by piece.

❸ ZOOMING

This is an example for the Series 3c only. The Series 5 does not have status windows and the Siena does not have large status windows owing to its screen size.

For each of the three types of status window, this program changes the font to implement zooming.

Press Psion-Z to cycle between small, medium and large fonts, and Shift-Psion-Z to cycle in the other direction. Esc changes to the next status window.

As well as changing the font and style for the text window (for PRINT etc.), the FONT command automatically changes the default graphics window size (ID=1) and the text window size to fit exactly in the space left by any status window. (A special feature not used here is that `FONT -$3fff,0` just changes the window sizes **without** changing font).

The procedure `dispinfo:` uses the command SCREENINFO to display the margin sizes in pixels between the default window and the text window, the text screen size in character units, and the text screen's character width and line height in pixels.

```
PROC tzoom:
   STATUSWIN OFF                       REM no status window
   zoom:                               REM display with zooming
   STATUSWIN ON,2                      REM large status window
   zoom:
   STATUSWIN ON,1                      REM and small
   zoom:
ENDP

PROC zoom:
   LOCAL font%(3),font$(3,20),style%(3)
   LOCAL g%,km%,zoom%
   zoom%=1
   font%(1)=13 :font$(1)="(Mono 6x6)" :style%(1)=0
   font%(2)=4 :font$(2)="(Mono 8x8)" :style%(2)=0
   font%(3)=12 :font$(3)="(Swiss 16)" :style%(3)=16
   g%=%z+$200
   DO
      IF g%=%z+$200
            IF km% AND 2               REM Shift-PSION-Z
                  zoom%=zoom%-1
                  IF zoom%<1 :zoom%=3 :ENDIF
            ELSE                       REM PSION-Z
                  zoom%=zoom%+1
                  IF zoom%>3 :zoom%=1 :ENDIF
            ENDIF
            FONT font%(zoom%),style%(zoom%)
            PRINT "Font=";font%(zoom%),font$(zoom%),
            PRINT "Style=";style%(zoom%)
            dispinfo:
            PRINT rept$("1234567890",15)
            gBORDER 0
      ENDIF
      g%=GET
      km%=KMOD
   UNTIL g%=27
ENDP

PROC dispinfo:
   LOCAL scrInfo%(10)
   SCREENINFO scrInfo%()
   PRINT "Left margin=";scrInfo%(1),
   AT 17,2 :PRINT "Top margin=";scrInfo%(2)
   PRINT "Screen width=";scrInfo%(3)
   AT 17,3 :PRINT "Screen height=";scrInfo%(4)
   PRINT "Char width=";scrInfo%(7)
   AT 17,4 :PRINT "Line height=";scrInfo%(8)
ENDP
```

ANIMATION EXAMPLE

This program requires five bitmap files - `one.pic` to `five.pic`. Each of these would differ slightly. They might, for example, be five 'snapshots' of a running human figure, each with the legs at a different point in their cycle.

The program copies each bitmap into a window of its own, then makes each window visible in turn, each time slightly further across the screen.

To make bitmap files, first draw the pattern you want with any of the graphics drawing commands. (Use `gLINEBY 0,0` to draw single dots.) When the pattern is complete, use gSAVEBIT to make the bitmap file. For advanced animation, you could use a sprite as described in the 'Using OPXs on the Series 5' section of the 'Advanced.pdf' document for the Series 5, and as described in the 'Advanced Topics' section of the 'Advanced.pdf' document for the Series 3c and Siena.

```
PROC animate:
   LOCAL id%(5),i%,j%,s$(5,10),w%,h%,edge%
   REM example width and height
   w%=16 :h%=28
   REM screen edge - use 480 for Series 3c and 240 for Siena
   edge%=640
   REM need not have ".pic" in the following for Series 3c and Siena
   s$(1)="one.pic" :s$(2)="two.pic" :s$(3)="three.pic"
   s$(4)="four.pic" :s$(5)="five.pic" :j%=1
   WHILE j%<6
      i%=gLOADBIT(s$(j%))
      id%(j%)=gCREATE(0,0,w%,h%,0)
      gCOPY i%,0,0,w%,h%,3
      gCLOSE i% :j%=j%+1
   ENDWH
   i%=0 :gORDER 1,9
   DO
      j%=(i%-5*(i%/5))+1            REM (i% MOD 5)+1
      gVISIBLE OFF                  REM previous window
      gUSE id%(j%)                  REM new window
      gSETWIN i%,20                 REM position it
      gORDER id%(j%),1              REM make foreground
      gVISIBLE ON                   REM make visible
      i%=i%+1 :PAUSE 2
   UNTIL KEY OR (i%>(edge%-w%))
ENDP
```

❸ TWO-VOICE “ICE-CREAM VAN” SOUND

This example is for the Series 3c and Siena only.

The following program plays a rising and falling scale. It uses the amplifier-driven loudspeaker device (with device driver SND:) which allows you to play tunes using two-note chords - ie it has two *voices*.

This program uses I/O keywords as described in the ‘Advanced Topics’ section. Take care to enter them exactly as shown here.

```
PROC main:
   LOCAL ret%,sndHand%
   ret%=IOOPEN(sndHand%,“SND:”,-1)     REM open the device
   IF ret%<0
      PRINT “Failed to start”
      PRINT err$(err)
      GET
   ELSE
      icecream:(sndHand%)
      IOCLOSE(sndHand%)
   ENDIF
ENDP

PROC icecream:(sndHand%)
   LOCAL notes1%(4),notes2%(14)
   LOCAL s1stat%,len1%,len2%
   REM define 1st voice
   notes1%(1)=1048 :notes1%(2)=96      REM freq, duration
   notes1%(3)=524  :notes1%(4)=48
   len1%=2                             REM number of notes in voice 1
   REM define 2nd voice
   notes2%(1)=1048 :notes2%(2)=16
   notes2%(3)=1320 :notes2%(4)=16
   notes2%(5)=1568 :notes2%(6)=16
   notes2%(7)=2092 :notes2%(8)=16
   notes2%(9)=1568 :notes2%(10)=16
   notes2%(11)=1320 :notes2%(12)=16
   notes2%(13)=1048 :notes2%(14)=48
   len2%=7                             REM number of notes in voice 2
   IOC(sndhand%,1,s1stat%,notes1%(),len1%)
   REM voice 1 asynchronous
   IOW(sndHand%,2,notes2%(),len2%)
   REM voice 2 synchronous
   IOWAITSTAT s1stat%
ENDP
```

`notes1%()` and `notes2%()` are set up to hold `len1%` and `len2%` notes to be played on voice 1 and voice 2 respectively. The number of notes to each voice must not exceed 16384.

Each note is composed of two consecutive integers in the array with the first of each pair giving the frequency in Hz (middle A is 440Hz) and the second giving the note duration in quarter-beats per minute.

INDEX