Pascal on the QL

by

A. F. Wilson

Unfortunately, Sinclair QL World (SQLW) magazine closed down in mid-1994 and the opportunity for this mini series on "Pascal for the QL" to be published in the magazine was gone. The unpublished preliminary mini series which has still to be checked is enclosed for posterity's sake.

CONTENTS

Pascal on the QL

Foreword

Choosing a programming language for a microcomputer is a time consuming and sometimes costly decision (e.g, ProPascal on the QL sells for £105.50). In the world of microcomputing four main languages dominate - Pascal, C , assembly language and Basic. During the 1980s, BASIC predominated because it was either inexpensive to buy or came free with most computers - CBM BASIC, BBC BASIC,ZX BASIC , MTX BASIC ,GFA BASIC , Hisoft BASIC ,GWBASIC, TurboBASIC , QuickBASIC and of course SuperBASIC. It is worth noting that ACE computers used FORTH instead of BASIC.

SuperBASIC is the gateway to the QL, much as Windows GEM and Workbench are today for the IBM PC, Atari ST and Commodore Amiga. The release of SuperBASIC was a milestone in the development of BASIC as a true programming language. It introduced superior control structures (ie, REPEAT - EXIT - END REPEAT) and procedures (i.e. DEFine PROCedure - END DEFine). Until this time, GOTOs and GOSUBs lead the BASIC programmer down a precarious path of spaghetti coding, difficult to read, amend and debug. However, unlike other languages which were standardising (ie, ANSI C), the BASIC language was left to develop without important industry definition on commands/syntax which ultimately makes BASIC less portable and less attractive to programmers.

Assembly language is a low level programming language and is very much processor (CPU) dependent (ie, 680x0 or 80x86 series). Although these two processors currently predominate the current market, a bit like the 6502 and Z80 of the 1970s and early 1980s, they will ultimately have a limited shelf life. Programming in assembly language is difficult as you have to write code from scratch, commands like INPUT and PRINT, etc. are just not readily available. A good knowledge of the operating system, i.e. QDOS ,is essential. Debugging assembly language code is a nightmare even with specialist debugging tools. C on the other hand shares some of the low level language advantages of operating system and hardware access and many of the advantages of high level languages. C code is fairly portable, however, it is difficult to read and debugging is also a bit tricking.

Pascal on the other hand is highly disciplined (structured) language and one of the main teaching languages in schools and universities. In many ways SuperBASIC is a subset of both Pascal and BASIC. Pascal forces a programmer to obey a fairly strict syntax and in some places is a bit to strict. However, the end result is worth the initial pain as the program is easy to debug with the help of over 400 different error messages (ProPascal), compared with a feeble 22 available in SuperBASIC. The source Pascal text is easy to read, because the code is structured and modular, changes can be easily accommodated with the minimum of effort. The code produced is not only fast and fairly compact but it offers a very high degree of portability, >80%, which is on par with C.

This series is not an 'Introduction to Pascal', there are plenty of books available that will more than adequately cover this, see the reference list. A visit to your local library or bookshop should be worthwhile. This series is a guide to Pascal from a Sinclair QL point of view. In this article, I will summarise the commands available to the Pascal programmer and where possible highlight the corresponding SuperBasic equivalent, this should provide a useful reference for those interested in converting programs from SuperBASIC to Pascal.

As there is a strong resemblance between SuperBASIC and Pascal , the transition from one language to other should prove fairly painless. Tables 1 to 9, (see appendix 1) provides a keyword match up of the

two languages. SuperBASIC commands listed do not include disc interface and Toolkit 2 extensions (see SQLW Series on SuperBASIC and its extensions). SuperBASIC commands specific to the BASIC interpreter and therefore not required for Pascal are load, lrun, save, mrun, merge, auto, edit, renum, list, dline, continue, retry, stop, run.

Those of you who take the time to browse through the tables will quickly realise that 3 or 4 key commands are missing - POKE_W/L ,PEEK_W/L ,DIR and FORMAT. The listings included in this series will rectify these omissions. DIR and FORMAT will also give us a chance at interfacing assembly language with Pascal. Each listing will generate a number of new Functions or Procedures which can be added to your library. I have written each test program in such a way as to demonstrate a particular Pascal command(s) (i.e. keyboard entry, file syntax, graphic commands, control structures, etc.). I hope you find Pascal as enjoyable as previous language series - C, LISP, SuperBASIC and Assembly language (the last two current to SQLW). Appendices 2 and 3 summarise the programs/utilities that come with ProPascal and how to use ProPascal with QPAC2. Appendix 4 provides a SuperBASIC to Pascal conversion program.

A.F.Wilson
Sept 1992

References:

SQLW, Feb. 1992, “Prospero Pascal” review by A.F.Wilson
Book “Writing Pascal Programs” by J S Rohl
Book “Mastering Pascal Programming”,by E.Huggins.
Book “Advanced QL Machine Code” by A.Denning
Book “The Sinclair QDOS Companion” by A.Pennel
QL User Jan. & Mar 1986 ,”Of Disks and Drives”,by C.Opie
SQLW series ,Oct. 1991-Jun. 1992,”Systematic Machine Code Programming” by A.Bridewell
SQLW, Mar 1992 - “Inside the C68 compiler” by S.Goodwin
SQLW, Dec 1992 - “Cport Review” by S.Butler
SQLW ,Sep 1990 ,Nov. 1990 and Feb. 1991, “Programming in C' by A.Wright and “C series” by A.Denning
PCW June 1989, “ANSI C” by N.Martin
PCW Jan.-May 1985, “Teach Yourself C” by L.Hampson
SQLW Oct 1990 "Internal Numbers on the QL" by S.Wallis
PCW "Pointing the way", by M.Hawes
PCW Aug 1988 "Maths on the double", by S.Dick
SQLW Sep 1989 "Archive Revealed" by D. Briggs,
SQLW Nov 1989 "The Archive Screen Driver" by M. Coulthard
"Psion Solutions" by R.Massey, Jun 1988 & H.Clase, April 1991
SQLW, vol 2, issue 7 “Very BASIC SuperBASIC" by D.Jones.
Manual “QL QPC, Concepts reference guide”, Tony Tebby / Sinclair Reserach Ltd / Miracle Systems

Pascal on the QL

1. SuperBASIC to Pascal

This series does not intend to be a 'How to Program in Pascal' book. If this is what you hoped for then I am afraid you best visit your local library. I decided to take a closer look at Pascal from a QL perspective and try to provide enough information for people to perform manual SuperBASIC to Pascal translations or vice versa, similar to the QL SuperBASIC to C utility, 'Cport'. The Pascal listings used throughout this manual were generated using the excellent Prospero software Pascal compiler - ProPascal. ProPascal is not only a superb implementation (see earlier Pascal reference) ,it is also available across a number of formats - Sinclair QL, Amstrad PCW, Atari ST, CP/M, PCDOS/MSDOS and OS/2. ProPascal conforms to the ISO standard and extensions (includes string and advanced file handling) and QL specific extensions (graphics). However, it is extremely expensive, although a bargain or two may be found in the classifieds (see Quanta, SQLW and MicroMart). On occasion, Pascal command extensions required assembly code. These were compiled using the excellent GST Macro Assembler.

This series attempts to cover in the broadest terms - SuperBASIC to Pascal translation ,Pascal commands and syntax, interfacing Pascal with Assembly language, Pascal extensions using QDOS (library building) and a number of useful Pascal utilities. To kick start this translation process, I have tabulated the main commands available in both SuperBASIC and Pascal, see appendix 1. I have included Pascal equivalents for as many of the universally accepted SuperBASIC extensions from Tony Tebby as found in Toolkit 2 (TK2) and the majority of disc interfaces. These tables are an essential reference during the translation process. I am sure a utility could be written to do this automatically, but that wouldn't be as interesting.

The keyword tables 1-9 (see appendix 1) do not include specific interpreter/compiler commands, e.g., for SuperBASIC - run, altkey, do, lrun, edit, clear, new, respr, continue, dline, load, save, local, auto, dlist, list, etc and for Pascal - FORWARD, EXTERNAL, PROGRAM, VAR, TYPE, CONST, {$p} , BEGIN, LABEL, COMMON, SEGMENT, PAGE, etc. Not all SuperBASIC commands have an exact equivalent, and rather than leave a blank, I have either written a procedure/function to mimic the SuperBASIC command. A typical example of a Pascal generated procedure to mimic the PAUSE command in SuperBASIC is:

```
PROCEDURE PAUSE(sbperiod:period24h);
    {period24h=4320000 for 50Hz machines was defined in the global TYPE declarations}
VAR sbcnt,pascnt:period24h;
BEGIN {pause}
    IF sbperiod=-1 THEN READLN  {await keypress}
    ELSE  {pause}
       Begin {else}
          FOR sbcnt:=0 to (period-1)  DO
          FOR pascnt:=1  to 2400 DO;
    End; {if-then-else}
END; {pause}
```

The syntax for 'PAUSE' in SuperBASIC is PAUSE <period> , where the period is defined as 1/50th of a second. The period is synchronised with the electricity frequency, which in the UK is 50Hz and in the USA its 60Hz. Therefore, in the UK a period of 50 is equivalent to 1 second. However, when I was testing the Pascal equivalent PAUSE(SBperiod); I found that the timings were completely different. In actual fact, the Pascal equivalent was 2400 times faster. To slow down the Pascal version, so that compatibility was retained with the SuperBasic period, a secondary loop using 'pascnt:=1 to 2400' was used.

In some instances, I have listed approximate Pascal equivalents , to point you in the right direction. The TK2 command 'PRINT USING' has no exact equivalent in Pascal, but, the Pascal command 'WRITE' offers a similar display output, (see tables 4 & 6 in appendix 1). For example, say you wanted to format the print output of the mathematical symbol PI (3.1415926) to 4 decimal places with no leading naughts or spaces. In Pascal this would be written as:

```
WRITE (output,'pi=',PI:6:4);
```

This command displays 6 characters to 4 decimal places as desired, e.g., 3.1415. You must include the full stop and any minus sign. See 'Mastering Pascal Programming', pg 31-32 by E.Huggins.

I apologise in advance for any translational errors. At this point, pick a SuperBASIC command a random, and lookup the translation tables to find the Pascal equivalent. Remember to check the SuperBASIC and Pascal manuals for syntax and definition. Initially, this part of the translation will be time consuming, however, translations will speed up with experience. Manual translations improve program understanding and this leads to better command selection, improves error correction and simplifies program enhancements.

There are a couple of useful utilities which list SuperBASIC variables and procedure/function names, i.e., Xref, Qref and BASIC Reporter. These utilities will greatly speed up translations by about 25% on programs greater than 100 lines and are therefore highly recommended. Translation times are very much dependent on the quality of the original program, for instance, structure and single line statements reduce translation times. This is where the program BetterBASIC helps you, it analyses your SuperBASIC program and corrects structural flaws, improves readability with indentation and reduces the number of multi-statement lines.

Listing 1 contains a number of Pascal functions/procedures to mimic some key SuperBASIC commands (i.e., POKE_W, POKE_L and the PEEK equivalents). The listings are contrived to highlight a specific command or programming technique within an environment familiar to SuperBASIC users. For instance, the SuperBASIC code:

```
IF (ch>='A') and (ch<='Z') then chkascii=1
else  chkascii=0
endif
```

or

```
a$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
if ch instr a$ then chkascii=1
else chkascii=0
endif
```

can be used to check if the character assigned to the variable 'ch' is a capital letter or not. In Pascal the above code would be translated as :-

```
IF (ch>='A') AND (ch<='Z') THEN chkascii:=true
ELSE chkascii:=false;
```

where the values 1 and 0 are equated to the predefined Boolean type with a range true to false. In Pascal you can redefine this further by taking advantage of the SET structure, refer to your Pascal manual and table 2. SETs are particular useful when working with lists, i.e., Sun..Sat, or Jan..Dec. The example below, checks if the character held in the variable 'ch' is a capital letter or not.

```
IF ch IN ['A'..'Z'] THEN chkascii:=true
ELSE chkascii:=fasle;
```

Finally, chapter 1 concludes by addressing some of those SuperBASIC commands without a Pascal equivalent. Listing 1 adds a number of new Pascal procedures and functions which are needed to mimic SuperBASIC. Type the listing using a text editor and save it to disc as testpeekpoke_pas. Now compile this will ProPascal. Assuming no errors, link the testpeekpoke_rel file with LINKER to generate testpeekpoke_bin. Now exec the program NOQNS to remove the annoying default screen prompts and your program is ready for executing (exec flp1_testpeekpoke_bin).

LISTING 1:

```
PROGRAM testpeekpoke(input,output);

  { Lines=  151   Code= 1824   Data=   36 }

CONST
    sv_ramt =163872;
    sv_caps =163976;
    kilobyte=1024;
TYPE
    longrange=0..2147483647;  { 32 bit }
   {nb:subset of integer, thus must obey integer type constraints}
    wordrange=0..65535;
    byterange=0..255;
VAR
    pokenum,peekaddr,pokeaddr,lastkey,ramsize:longrange;
    ch:char;
    i:1..10;
    buf1:string[10];
    ascii:boolean;

{******** User Functions ********}

{ SuperBASIC keyword: peek_W }
FUNCTION  peek_W(peekaddr:longrange):longrange;
{ numbers stored in MSB/LSB format in RAM. }
Begin
   peek_W:=peek(peekaddr+1)+(peek(peekaddr)*256);
end;  { peek_W }

{ SuperBASIC keyword: peek_L }
FUNCTION peek_L(peekaddr:longrange):longrange;
{numbers stored in RAM in MSB/LSB format}
{(addr+3*2^0) + (addr+2*2^8) + (addr+1*2^16) + (addr+0*2^24)}
begin
  peek_L:=peek(peekaddr+3)+(peek(peekaddr+2)*256)+
              (peek(peekaddr+1)*65536)+(peek(peekaddr)*16777216);
end;  { peek_L }

{ SuperBASIC keyword: poke_w }
PROCEDURE  POKE_W(pokeaddr:longrange;pokenum:wordrange);
{ numbers stored in MSB/LSB format in RAM. }
CONST
  Lmsb=256;
  Llsb=1;
```

```
VAR
  p0:byterange;
Begin
  p0:=trunc(pokenum/Lmsb);
  poke(pokeaddr+0,p0);
  poke(pokeaddr+1,(pokenum-p0*Lmsb));
end;  { POKE_W }

{ SuperBASIC keyword: poke_l }
PROCEDURE POKE_L(pokeaddr:longrange;pokenum:longrange);
{ numbers stored in MSB/LSB format in RAM. }
CONST
  Hmsb=16777216;
  Hlsb=65536;
  Lmsb=256;
  Llsb=1;
VAR
  p0,p1,p2:byterange;
Begin
  p0:=trunc(pokenum/Hmsb);
  pokenum:=pokenum-(p0*Hmsb);
  p1:=trunc(pokenum/Hlsb);
  pokenum:=pokenum-(p1*Hlsb);
  p2:=trunc(pokenum/Lmsb);
  pokenum:=pokenum-(p2*Lmsb);
  poke(pokeaddr+0,p0);
  poke(pokeaddr+1,p1);
  poke(pokeaddr+2,p2);
  poke(pokeaddr+3,pokenum);
end;  { POKE_L }

FUNCTION uppercase(ch:char):char;
begin
  if (ch>='a') and (ch<='z') then
            uppercase:=chr((ord(ch)-ord('a'))+ord('A'))
  else
     uppercase:=ch;
end;

FUNCTION lowercase(ch:char):char;
begin
  if (ch>='A') and (ch<='Z') then
            lowercase:=chr((ord(ch)-ord('A'))+ord('a'))
  else
     lowercase:=ch;
```

```
end;

FUNCTION chkascii(ch:char):boolean;
begin
  if (ord(ch)>31) and (ord(ch)<128) then chkascii:=true
  else
     chkascii:=false;
end;

{**Graphics extensions to PASCAL - machine dependent (QL - QDOS)**}

{SuperBASIC keyword:  MODE}
PROCEDURE mode(highres: boolean);  EXTERNAL;

{SuperBASIC keyword:  WINDOW}
PROCEDURE window(VAR w: text;
                         width,height,Xorigin,Yorigin: integer);  EXTERNAL;

{SuperBASIC keyword:  CLS}
PROCEDURE cls(VAR w: text;part: integer);  EXTERNAL;

BEGIN {main program - test the above procedures/functions}
mode(true);                                  {Monitor mode }
window(output,420,160,20,20); {redefine/reposition default window}
cls(output,0);                          {cls whole window}

ramsize:=PEEK_L(sv_ramt);
writeln(output,'QL Ram Size is ',trunc(ramsize/kilobyte):4 ,' K ');
writeln(output);

POKE_W(sv_caps,65280); {set caps on}
write(output,'CAPS LOCK now on,type some letters (10 max)
            and see:  ');
readln (input,buf1);
writeln(output);

write(output,'The LOWECASE function converts the above line to: ');
for i:=1 to 10 do  write(output,LOWERCASE(buf1[i]));
writeln(output);

write(output,'The UPPERCASE function converts back again:' );
for i:=1 to 10 do write(output,UPPERCASE(buf1[i]));
writeln(output);

ascii:=CHKASCII(buf1[3]);
```

```
if ascii=true then writeln(output,'The third letter is an ASCII character')
else writeln(output,'The third letter is not an ASCII character');
writeln(output);

writeln(output,' Press ENTER to exit ');
readln(input);
END.
```

Pascal on the QL

2. Parlez Vous Pascal?

The last section focussed on SuperBASIC to Pascal command translations. However, it wasn't possible to directly translate every command. A number of powerful additions were generated in listing 1 to minimise this shortfall. This chapter will continue this process, in particular, attempt to address the very powerful DATA, RESTORE, READ SuperBasic structure. A SQLW magazine favourite, 'Hex Loader v3' by M.Jeffry & S.Goodwin (listing 2), was translated, see listing 3, both listings can be found at the end of this section. "DON'T PANIC" as Douglas Adams wrote in his book 'Hitchhikers Guide to the Galaxy'. The doubling of the source text (e.g., extra 65 lines) is easily explainable:

(1) Pascal's need for the BEGIN and END identifiers when using the following commands - WHILE, FOR, IF, CASE, RECORD, PROCEDURE and FUNCTION is not not fully implemented in SuperBasic. SuperBASIC never uses begin, however, END PROCedure, END FUNCTion, END IF, END SELect, END REPeat and END FOR are used. NB: Pascal's UNTIL and SuperBasic's END REPeat statements are assumed to be equivalent. In this example, this difference alone accounts for 11 of the 65 lines (17%).

(2) The Pascal version uses mostly single statement lines compared with multi-statement lines used in SuperBasic, e.g., SuperBASIC line 150 can be re-written as 4 single statement lines -

```
150 CLS
152 RESTORE
154 READ space
156 start=RESPR(space)
```

When translating, single statement lines are preferred. BetterBasic utility will do this for you. Converting the SuperBASIC listing into "one liners", adds 11 extra lines to the SuperBasic listing (17%).

(3) As discussed in the earlier chapter, Pascal programming places a lot of emphasis on variable declaration (CONST/TYPE/VAR). This fundamental approach to variables forces Pascal programmers to question the need, use and value of each variable. It is this good programming practice (GPP) which makes Pascal so powerful and ideal as a training tool. In this example, variable declaration added an extra 22 lines compared with the SuperBASIC version (34%).

(4) Pascal Program title and library extensions added another two lines (3%). The inclusion of the graphics library {$I flp1_graphics_pas} allows window operations if required. In SuperBASIC this is used implicitly, whereas, this has to be included in Pascal. This inclusion/exclusion of libraries can greatly reduce the program overhead, execution speed and improves the portability to other computers. The portability issue will be discussed length in the next chapter.

(5) SuperBASIC commands with no direct Pascal equivalents, e.g., the powerful DATA, RESTORE, READ command set. This equates to the remaining 19 lines (29%).

From this you could say that the portability from SuperBASIC to Pascal is about 70%, which isn't bad considering that 70-80% is considered the normal for Pascal to Pascal or C to C conversions across computer platforms. The extensions made to Pascal in this manual will increase the translation or %portability to nearer 80-85%.

The importance of both variable declaration and the DATA, RESTORE and READ command set require special attention.

Variables

The SuperBASIC interpreter recognises the following 4 default data types:

a. Real with a lower to upper limit of E-615 to E+615.

b. Integer (%) with a lower to upper limit of -32768 to 32767.

c. String ($) with a lower to upper limit of 1 to 255 characters.

d. Array (DIM), this is dependent on the dimension of the array (e.g., 3 dimensional) and the declaration of the dimensions data type (e.g., real, integer or string).

The following rules are applied during variable declaration:

e. All variable names must start with a letter (a to z or A to Z). The rest of the variable name can then be followed by alphanumeric characters. For example, *de34=10* or *alpha=4/2* are both valid whereas *3de* is not acceptable.

f. The SuperBASIC interpreter assumes that all variable name assignments are real (default data type). To change the data type to integer or string requires appending the following identifiers *%* for integer and *$* for string to the variable name. For example, integer variable names *of a%* or *dec10%* are both valid, whereas, *getthis$* would be a typical string variable name.

g. The assignment of values to variable names must be type compatible and be within the specified lower and upper limits, see below.

Valid real assignments are *a=10* or *alpha=109.456* or *d2=1e-7*, whereas *a="C"* is not valid.
Valid integer assignments are restricted to non-fractional number values, e.g., *a%=100*, but the assignment *a%=100.1* is not permitted, as 100.1 is a real value. Both integer and real data types are restricted to numerical assignments, whereas, the string data type can be alphanumeric (numerical and/or characters).
The string data type in SuperBASIC (or any BASIC implementation) is one of the most

important features of the language. The default length of a string is 255 characters, however, using the free field format style, the string variable can either shrink or expand, depending on the assignment, as long as the maximum is not exceeded, e.g. the string length is 5 for the assignment, *greek$="alpha"*,sometimes written as *LET greek$="alpha"*. The string decreases in length to 4, when *greek$="beta"*.

The array data type is the only one of the four which requires declaration prior to assignment e.g.

```
10 DIM a(10))
20 a(3)=1.789e-7
```

For example, a three dimensional array of data type real is declared as

```
100 DIM a(10,20,5).
```

All SuperBASIC variables are GLOBAL by default, however, within procedures and functions using the command LOCal, a variable can be declared before assignment. The local variable cannot be accessed/changed out with the procedure or function and therefore are easier to manage. Global variables on the other hand, are assigned as and when needed and if not carefully monitored can lead to mis-assignment and increases the likelihood of errors. Therefore to summarises, SuperBASIC local variables have to be declared and then assigned, whereas, global variables are just assigned, except for array's which require prior declaration.

Pascal forces programmers to THINK about the use of ALL variables.

Pascal requires a detailed understanding of the uses of the variable prior to any declaration or assignment. Pascal forces you at an early stage to decide whether the variable is used for controlling a loop, or used in a calculation or used for byte operations, etc. You must then decide the data type, its range and whether its a CONST, global or local variable. For example, if you want an integer variable to be restricted to the range 1 to 10, use the declaration:
VAR num:1..10;

If you inadvertently assigned num in the main source text as 100, e.g., *num:=100;* then an error would be reported during compilation, e.g. 'Range error at line ??'. However, in SuperBASIC, *num%=100*, does not flag up any error on input, resulting in runtime problems, which are often difficult to track down. This is because all SuperBASIC integer variables can be assigned to any value in the range -32768 to 32767, before an error is reported.

This is an example of Pascal's superior error trapping at compile time. Pascal is also very efficient at trapping runtime errors. Extending the above example to include the line
num:=num+1;
,highlights a common runtime error. In Pascal, if num exceeded 10 in the course of runtime, this will result in a runtime error (eg. 'Bounds exceeded'), however, in SuperBASIC, there is no built-in error trapping of this type and num% could end up at 1000. An error like this could have

caused unrecoverable damage, especially if num% was involved in file I/O operations.
This type of error is easily avoided in SuperBASIC using either a FOR/NEXT loop or control statement, eg. IF num%>10 THEN ... .The above trivial example does highlight potential pitfalls which are not easily debugged mainly due to the fact that the problem usually manifests itself in another part of the program. Pascal's error reporting is extensive with over 400 compile/runtime error messages compared with a feeble 22 in SuperBASIC.

Pascal takes the definition of variables a stage further in its structured approach to good programming practice. As shown earlier, in the variable declaration - VAR - the variable name is defined in terms of the subrange, ie *num:1..10;* or *num:51.102;* etc. The TYPE definition assigns a name to the variable range. For example,

```
TYPE byterange=0..255;
VAR   bytenum:byterange;
```

is equivalent to VAR bytenum:0..255;

The TYPE declaration is necessary for PROCEDURE declarations,eg.

PROCEDURE getbyte(bytenum:0..255); is not permitted in ProPascal

whereas

PROCEDURE getbyte(bytenum:byterange); is permitted.

In Pascal, there are two types of variable:

(1) variables that remain unchanged throughout runtime, declared as CONST. For example,

CONST pi=3.14159;

As already stated, once a variable is declared as a CONSTant, its assignment cannot change during runtime. The example above does not permit the following assignments or operations; *pi:=3.1415926;* or *pi:=pi+1;* , however, *x:=pi+1;* is permitted as pi remains unchanged.

(2) variables that are assigned different values throughout runtime, declared as VAR or TYPE/VAR, see above.

Standard Pascal variables can be declared as boolean, real, integer, string, array, characters, records, set, file, text, pointers, longreal and subranges thereof. The table 9 in appendix 1 lists the main data type ranges and the RAM requirements for holding the variable assignment, e.g., *adat:=10.1;* requires 4 bytes of RAM to store the REAL value 10.1. All REAL values are stored in RAM in floating point notation. This notation requires a minimum of 4 bytes to hold the relevant information. For reference read the following articles - "Internal Numbers on the QL" by S.Wallis, SQLW Oct 1990 ; "Pointing the way", by M.Hawes, PCW and "Maths on the double",

by S.Dick,PCW Aug 1988.

A number of the above data types ranges can be shortened (i.e., subranges) depending on the application and at the same time reduces the RAM storage requirements to 1, 2, 4 or 8 bytes. For example, if bytenum was declared as 0..255 (i.e., integer data type), then the RAM requirement is only 1 byte. However, if the subrange of bytenum was declared as 32768..65535, then the RAM requirement is 2 bytes, and so on.

Before moving on, it is worth noting, the more bytes used to store a REAL value, can either increase the range or increase the precision (accuracy) of the stored floating point number. Rounding errors are minimised if you increase the precision. The QL uses 6 bytes to store all REAL values in the range E-615 to E+615 up to 8 significant places (or 8-bit precision). Pascal compromises the range of data types REAL and LONGREAL to give greater precision, see table 2.1.

The declaration and assignment of a Pascal variable is therefore highly structured and requires careful thought. This emphasis on the declaration and assignment phase in Pascal programming is very important for the following reasons-

1. Encourages greater thought about the purpose of all variables and what controls will be needed (subrange definition)
2. Reduces errors (at compile & runtime)
3. Minimise RAM requirements (subranges)
4. Improves readability (variables and data types at start)
5. Speeds up changes (see 4)

DATA, RESTORE and READ

In SuperBASIC, the DATA structure is defined as:-

<linenum> DATA field1 (,field2,field3,etc)

for example,

590 DATA 140
750 DATA '4A804E7570FA4E750000','*',11006

In Database terminology, the <linenum> is unique and therefore equivalent to the record number, and the numbers and/or text preceding the DATA keyword, separated by commas, define the number of fields within the record, the data type for each field and the RAM requirements to store the data. The fields require no declaration and can be of variable length, i.e. free field format. Although this simplifies programming, the burden is shifted to the SuperBASIC interpreter, which then has to second guess the programmers usage (artificial intelligence!!), in order to determine the field structure, and all this during runtime, obviously the penalty is program speed.

For example, line 590 is a record of only 1 field, however, the data type and therefore the RAM allocation is ambiguous, as the number 140 can be either an integer or real value. In this example, the variable name assigned to READ the data value confirms the actual data type. The SuperBASIC command *READ space*, confirms that the data type was of real. However, If *READ space%* had been used instead, then the data type would have been of integer. In actual fact, *space%* would be more appropriate, for two reasons - (a) you cannot reserve fractional RAM with the RESPR command and (b) only 2 bytes are needed to store *space%* in RAM compared with 6 bytes for *space*.

Line 750 is an example of a multi-field data structure and as before the data type associated with each field is specified using the *READ var*. Each field is separated with commas (,) and all non-numeric fields (e.g., String data) are either enclosed with ' ' or " ". To assign this data to a SuperBASIC variable, as you expected, is no different from the single field case, either *READ field1: READ field2: READ field3* or *READ field1,field2,field3*.

I interpreted the DATA between lines 590 and 750 as numeric data, e.g., the machine code space requirements (140) and the check total (11006), as distinct from the machine code text. Therefore, both these figures should be separated from the main data, e.g.

590 DATA 140: REM space requirements
600 DATA 11006 :REM check total

Using *RESTORE: READ space:READ* check to assign the data to the correct variable.

Lines 610 to 740 would remain unchanged, however, line 750 would be rewritten, see line 600

and below:-

```
750 DATA '$A80.....000'
760 DATA '*':REM end of data
```

The above new format also simplifies the structure and eases the translation into Pascal. The new machine code text structure is one field of data type string, 40 characters in length, holding machine code text. The SuperBASIC variable used to read this data is *h$*. h$ is also used to read the end marker '*'. The end marker is used to exit the *load_hex_digits loop* (see, lines 320 to 470). Therefore, logically, line 760 is valid. The data between lines 620 and 760 are assigned to h$ sequentially using the *READ h$* command. Please note that the SuperBASIC interpreter, performs the linenum increment automatically after every READ, so that the DATA pointer is at the correct data.

Those of you who read chapter 1, will have read my comments on multi-statement lines and good structure. Quite clearly, some pre-translation and understanding of the SuperBaSIC program is needed to help conform the SuperBASIC listing into a format which simplifies translation, as in the case above. Fortunately, the above authors scored well into the 90s for structure, this is not usually the case.

If you compare listing 2 and 3 with respect to the data, you will see that I used two CONSTants for *space* and *check*. The Pascal procedure *read_data(hstr,linenum)* is equivalent to the SuperBASIC commands - *RESTORE linenum:READ h$*. The procedure read_data, uses the linenum to select the correct data to assign to hstr. The control structure CASE, is equivalent to SELect in SuperBasic, see the article "Very BASIC SuperBASIC" by D.Jones, SQLW, vol 2, issue 7. The CASE command is a powerful feature of Pascal.

I have tried to use similar SuperBASIC keywords in the Pascal version. Although Pascal supports two commands - READ and READLN, these are not in any way connected to the SuperBASIC READ command. In actual fact, they are equivalent to the INPUT command. Similarly, WRITE and WRITELN in Pascal are equivalent to PRINT. When in doubt checkout tables 1 to 8 in appendix 1. Therefore, instead of READ variable, the Pascal version was *read_data(variable,linenum)* . By assigning *hstr:='43FA.....'*, we are just writing out in full what the SuperBASIC interpreter performs invisibly, each time it READs the DATA. The linenum was required to retain compatibility with the SuperBASIC command *RESTORE linenum*. The next chapter extends the read_data procedure to handle multiple field DATA.

Other Translation Issues

Pascal does not support RESPR, however, RESPR is simply a DIM statement for storing SuperBASIC subroutines in a reserved RAM area, distinct from normal RAM. Therefore, during conversion, I have substituted an ARRAY for RESPR and the CONSTant space controls the upper limit of this array.

Pascal, does not like complex statements like line 210, which includes an IF clause. In Pascal, the IF clause cannot be included in the assignment and therefore requires a minor modification, see FUNCTION decimal. Pascal supports both CODE as ORD and CHR$ as CHR, check manual for syntax, etc.

The MOD syntax for SuperBASIC (line 330) is almost the same as in Pascal. The function of Line 330 is to check if the machine code string held in h$ is odd or even. This test function is better performed in Pascal, using the function ODD (syntax is *ODD(true)* or *ODD(false)*), without having to test for division remainders>0 to confirm odd or even.

Pascal, does not support a STOP equivalent, although this can be achieved using the LABEL/GOTO structure, please refer to your Pascal manual. I prefer not to jump out of loops/procedures abruptly like this and usually, use a errflag or stopflag to monitor status. This conversion ,uses the errflag and the linenum to pinpoint any problem line(s).

Listing 2 uses a simple file handling structure - enter file name (INPUT) and then save the binary data to disc (SBYTES). As already stated the Pascal equivalent of INPUT is READ or READLN. The SuperBASIC INPUT command includes a print prompt capability, e.g. *150 INPUT 'Save to file...';f$* . This could be rewritten as *PRINT 'Save to file...';:INPUT f$* . Pascal, uses the latter approach - *WRITE('Save to File');READLN (fstr);* . The Pascal translation of f$ is fstr, however, I prefer to use the variable name, qlfn (acronym of ql filename) for filename I/O.

In Pascal, the filename must then be assigned to a device, e.g. *ASSIGN(fout,qlfn);* . The device, in this case, is fout, which was declared as a FILE of mcodetype (i.e. machine code or binary format), to maintain compatibility with SBYTES. Once assigned, the binary data held in the Pascal array, start, can be written to this device, ie. saved to disc as a binary file, using the Pascal commands - *REWRITE(fout); FOR l:=1 TO byte DO WRITE(fout,start[l]); CLOSE(fout)*, which is equivalent to the SuperBASIC command *SBYTES f$,start,byte* .

The problem with Pascal, is that the FOR-END {for} loop is limited in its use compared with the SuperBASIC commands FOR-NEXT and FOR-EXIT-END FOR with STEP extensions. How increments or steps of >+1 or <- 1 were overlooked is one of those great unresolved questions. So much detail was placed on everything else, maybe this was just overlooked in all that excitement!! Anyway, line 370 has no direct equivalent in Pascal, instead the programmer has to resort to a step counter inconjunction with either the REPEAT - UNTIL loop or the WHILE-DO-END loop. Both loop structures are equally valid, depending on the function of the loop. However, there is a fundamental difference in the REPEAT - WHILE methodology.

The WHILE loop requires certain conditions to be met before the code within the loop can be executed, whereas, a REPEAT loop, the code is executed once before conditional testing. In this case, b was used as the step counter, as used in the SuperBASIC version. The step counter was incremented in steps of two, using the Pascal statement *b:=b+2;* . In the SuperBASIC version, the FOR loop variable, b, is automatically incremented by 2. In this translation, the while loop was selected, see *hex_load* procedure, listing 3.

LISTING 2: SQLW magazine favourite, 'Hex Loader v3' by M.Jeffry & S.Goodwin

```
100 REMark QL World Hex Loader v 3
110 REMark by M Jeffrey & S Goodwin
120 :
150 CLS:RESTORE : READ space: star
    t=RESPR(space)
160 PRINT 'Loading Hex...' : HEX_L
    OAD start
170 INPUT 'Save to file...';f$
180 SBYTES f$,start,byte : STOP
190 :
200 DEFine FuNction DECIMAL(x)
210 RETurn CODE(h$(x))-48-7*(h$(
    x)>'9')
220 END DEFine DECIMAL
230 :
240 DEFine PROCedure HEX_LOAD(start)
290 byte = 0 : checksum = 0
300 REPeat load_hex_digits
310    READ h$
320    IF h$='*' : EXIT load_hex_di
    gits
330    IF LEN(h$) MOD 2
340       PRINT'Odd number of hex
    digits in: ';h$
350       STOP
360    END IF
370    FOR b = 1 TO LEN(h$) STEP 2
380       hb = DECIMAL(b) : lb = DEC
    IMAL(b+1)
390       IF hb<0 OR hb>15 OR lb<0
    OR lb>15
400          PRINT'Illegal hex digit
    in: ';h$ : STOP
420       END IF
430       POKE start+byte,16*hb+lb
440       checksum = checksum
     + 16*hb + lb
450       byte = byte + 1
460    END FOR b
470 END REPeat load_hex_digits
480 READ check
490 IF check <> checksum
500    PRINT'Checksum incorrect.
    Recheck data.':STOP
520 END IF
530 PRINT'Checksum correct, data
    entered at: ';start
560 END DEFine HEX_LOAD
570 :
580 REMark Space requirements
    for the machine code
590 DATA 140
600 :
610 REMark Machine code data
620 DATA '43FA000834780110 4ED2'
630 DATA '0001001006535F464F4E'
640 DATA '540000000000000B7CD'
650 DATA '67064A33E8016B0472 01'
660 DATA '60222A0D4BEB00083478'
670 DATA '01124E924A8067047 0F1'
680 DATA '4E75264D2A453231E800'
690 DATA '54892D4900 58C2FC0028'
700 DATA 'D2AE0030B2AE00346C2C'
710 DATA '203618006B2620403478'
720 DATA '01184E924A8066CE5343'
730 DATA '6708534366C62471E804'
740 DATA '2271E800702576FF4E43'
750 DATA '4A804E7570FA4E750000','*',11006
```

LISTING 3: Listing 2 converted to Pascal.

```
PROGRAM hex_loadv3p;
{SuperBASIC Version of QL, World Hex Loader v3 by M.Jeffry & S.Goodwin}
{Pascal conversion by A.F.Wilson}

CONST
        space= 140; {space requirements for the machine code}
        check= 11006; {machine code checksum total}

TYPE
        hstrtype=string[40];
        lnurnrange=1..99999;
        mcodetype= 0..255;
        qlfntype=string[35];
        resprtype= array [1..space) of mcodetype;

VAR
        fout:file of mcodetype;
        qlfn:qlfntype;
        hstr:hstrtype;
        linenum:lnumrange;
        mcode:mcodetype;
        l,byte:integer;
        start:resprtype;
        errflag:boolean;

{$I flp1_graphics}

PROCEDURE read_data(var hstr:hstrtype;linenum:lnumrange);
begin {read_data}
  case linenum of
        620: hstr:='43FA0008347801104ED2';
        630: hstr:='0001001006535F464F4E';
        640: hstr:='5400000000000000B7CD';
        650: hstr:='67064A33E8016B047201';
        660: hstr:='60222A0D4BEB00083478';
        670: hstr:='01124E924A80670470F1';
        680: hstr:='4E75264D2A453231E800';
        690: hstr:='54892D490058C2FC0028';
        700: hstr:='D2AE0030B2AE00346C2C';
        710: hstr:='203618006B2620403478';
        720: hstr:='01184E924A8066CE5343';
        730: hstr:='6708534366C62471E804';
```

```
        740: hstr:='2271E800702576FF4E43';
        750: hstr:='4A804E7570FA4E750000';
        760: hstr:='*';
  end; {case}
end; {read_data}

FUNCTION decimal(x:integer):integer;
begin
        if hstr[x]>'9' then decimal:=ORD(hstr[x])-48-7
                else decimal:=ORD(hstr[x])-48;
end;

PROCEDURE hexload(var start:resprtype);
VAR
        hb,lb:mcodetype;
        b,checksum:integer;

begin {hexload}
        byte:=1; checksum:=0;
        linenum:=620; errflag:= false; {new variables}
        Repeat
                read_data(hstr,linenum);
                If hstr[1]<>'*' Then
                begin
                        if (length(hstr) mod 2)>0 then
                        begin {then}
                                errflag:=true;
                                writeln(output,'Odd number of hex digits at linenumber ',linenum);
                        end {then}
                        else
                        begin {else}
                                while ((b<=length(hstr)) and (errflag=false)) do
                                begin {while}
                                        hb:=decimal(b);
                                        lb:=decimal(b+1);
                                        b:=b+2; {increments of 2}
                                        if ((hb<0) or (hb>15)) or ((lb<0) or (lb>15)) then
                                        begin {then 2}
                                                errflag:= true;
                                                writeln(output,'Hex digit error at linenumber
',linenum);
                                        end {then 2}
                                        else
                                        begin {else 2}
                                                start[byte]:=16*hb+lb;
```

```
                                        checksum:=checksum+start[byte];
                                        byte:=byte+1;
                                    end; {else 2}
                                end; {while}
                            end; {else}

                            linenum:=linenum+10;
                        end {if}
                Until (hstr[1]='*') or (errflag= true);

                if checksum<>check then
                begin
                        errflag:=true;
                        writeln(output,'checksum incorrect. Recheck data. ');
                end;
end; {hex load}

BEGIN {main}
        cls(output,0);
        writeln(output,'Loading Hex ....');
        hexload(start);
        if errflag= false then
        begin
                write(output,'Save machine code file as ');
                readln(input,qlfn);
                assign(fout,qlfn); rewrite(fout);
                for l:=1 to byte do write(fout,start[l]);
                close(fout);
        end;
END.
```

Pascal on the QL

3. Perfectly Pascal

Extending the SuperBASIC DATA,RESTORE, READ structure in Pascal to include multiple fields, see listings 4 and 5, requires a basic understanding of the structure of a database. A database is a glorified table made up of a series of columns (i.e., fields) and rows (i.e., records) containing data, see figure 1. These tables are managed in terms of table design, field type (e.g., numeric, character, date, etc.), sorting, reporting, etc. by either the database command menus or user generated database programs. Archive on the QL is such an application which allows both direct command control of these tables or indirect control via user designed program code using the Archive language. Database languages like that of "dbase" for PC compatibles are either based on Pascal or Basic, containing extensions like sorting, indexing, etc.

Therefore, it comes as no surprise when I say that converting Archive listings to Pascal is similarly possible, although specific sorting, searching, indexing, etc. functions and procedures will have to be generated. This is not a problem as the majority of the "How program in Pascal" books do cover these topics. The SuperBASIC DATA, RESTORE, READ structure is a very simplistic database table manager which is very flexible, particularly for small amounts of data. I intend to develop the single field data translation introduced earlier, see listings 2 and 3.

Multiple fields within a SuperBASIC DATA command are simply a list of data separated by commas, e.g.,

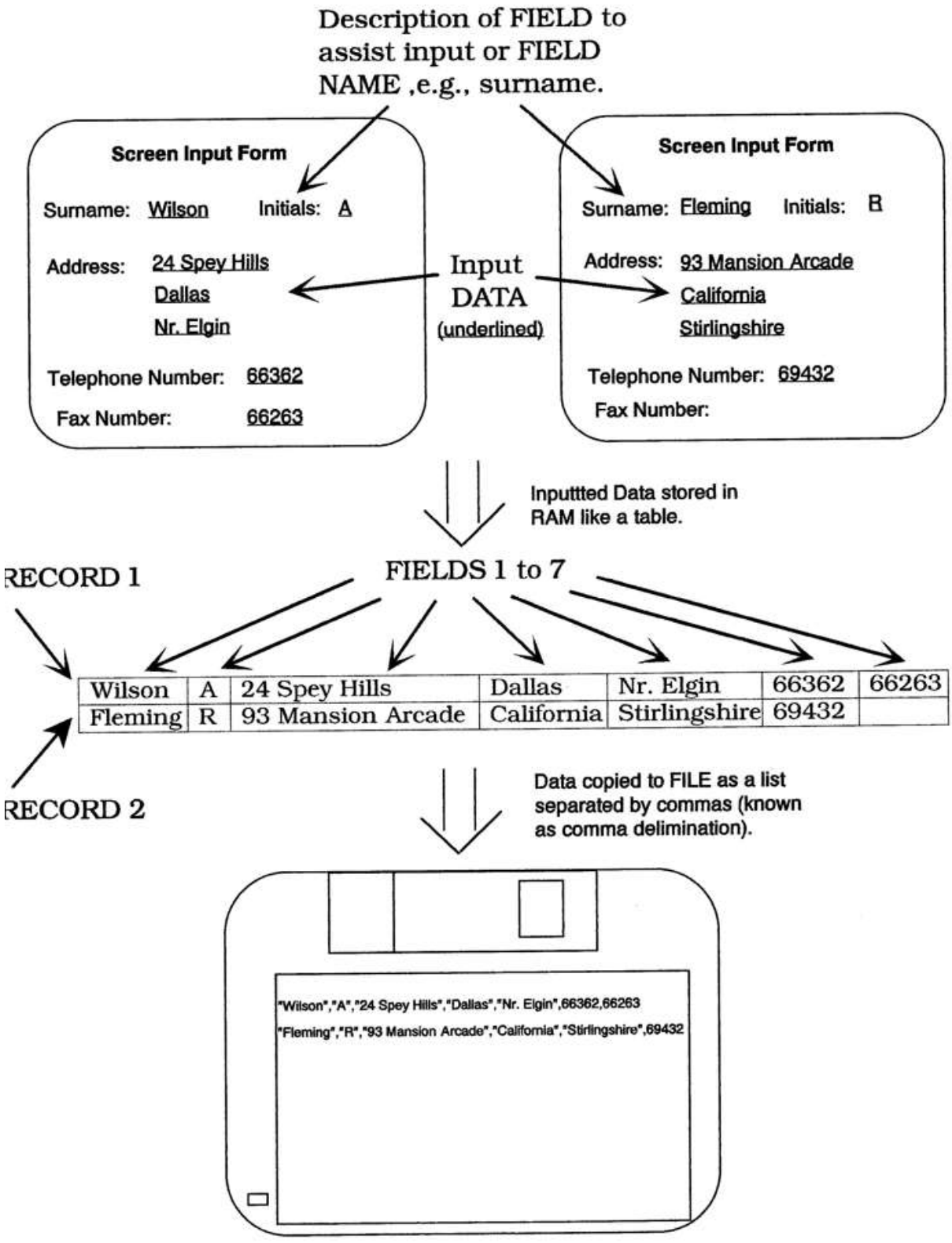
```
100 DATA 10,20,30.5,"seconds"
```

This statement defines 4 fields, with field1=10, fields2=20, field3=30 and field4=seconds. The first 2 fields can either be of integer or real type, field3 is definitely of real type and the later is a string, see previous chapter for more detail. The SuperBASIC interpreter at this stage probably assumes the data to be either REAL or STRING ($) type. Only when the SuperBASIC interpreter converts the READ command into assembly language will it know for certain the exact type, e.g.,

```
500 RESTORE 100: READ run%,temp,time,tunits$
```

The above instructs the SuperBASIC interpreter to find the above data in memory with a line number reference of 100 and to treat field1 of the listed data as an integer type instead of real. The SuperBASIC interpreter only knows the data type on executing the READ variable list.

Figure 1 - Database Outline



While the listing size is concise, the lack of pre-processing or compiling greatly reduces the speed performance of the program. This will not happen in Pascal. The Pascal *read_data* procedure in listing 3 will now be extended to address multiple field data lists. Pascal already contains the core elements of a database with a special type structure called RECORD which is made up of fields, e.g.,

```
Type
        Exptrec= RECORD
                        run:integer;
                        temp:real;
                        time:real; {some Pascal implementations include a special time type}
                        tunits:string[7];
                END;

Var
        expt:exptrec;
```

In a Pascal program the field information would be accessed via the Pascal statement:

```
expt.run:=10;
```

the full stop separates the record from the field names. The above example can only handle one record. Changing the VAR definition to *expt:packed array[1.100] of exptrec;* would hold 100 records, just like a database. To access the different records is like working with an array incorporating the field sub-assignment, e.g., *expt[10].tunits:=minutes;*

Please consult your Pascal books/reference manuals for more details.

EPSON VIEW

It is now time to conclude this brief look at SuperBASIC to Pascal translation, by detailing the new read_data procedure, and then looking at file, printer, and window translations. Fortunately, listing 4 already encompasses all these features and at the same time covers the QL past-time of Printer Drivers. Over the years, printer driver questions, utilities, tips, etc. have been covered extensively in all QL related publications, e.g., Quanta and SQLW. Only recently, SQLW published a HP deskjet utility, see SQLW, vol 2, issues 5&6.

At the outset, I had decided to reduce the number of SuperBASIC variables declared globally by including them within the appropriate procedures, thus improving legibility and minimising errors. A conscious decision was made to translate listing 4a with portability and upgrade-ability in mind, see later.

Listing 4a is a cut-down version of a dot matrix printer emulating the EPSON control code command set, i.e., ESC,E,0 or 27,69,0 to set the printer to BOLD. The procedures - prt_dev, read_data_ptf, and send_prt_msg can be easily modified to provide DEC VT52 screen/terminal

emulation or PSION Archive screen emulation, i.e., SQLW articles - "Archive Revealed" by D. Briggs, Sep 1989; "The Archive Screen Driver" by M. Coulthard, Nov 1989 and "Psion Solutions" by R.Massey, Jun 1988 & H.Clase, April 1991.

During the translation I took the opportunity to rewrite the menu system (not included in Listing 4a) to improve the layout, and include more options, i.e., inclusion of typeface commands like Bold_on or Italic_off, etc. This modification resulted in an extra field which was linked with the appropriate Epson control code, e.g., **Bold_on** would be translate into **27,69,0**.

The SuperBASIC listing stored the Epson printer control codes like a table consisting of 9 rows of 3 columns, see figure 2a. In SuperBASIC this was written as *DIM ptfcode%(9,3)*. This array expects data of type integer as indicated by the **%** designator. Procedure *init_epson_ptfaces* describes how to assign data to the array.

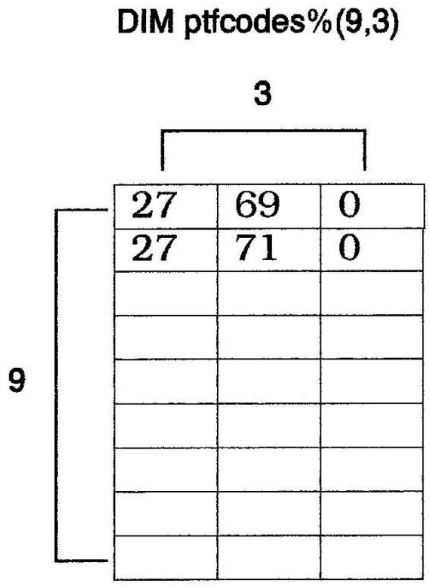


FIGURE 2a

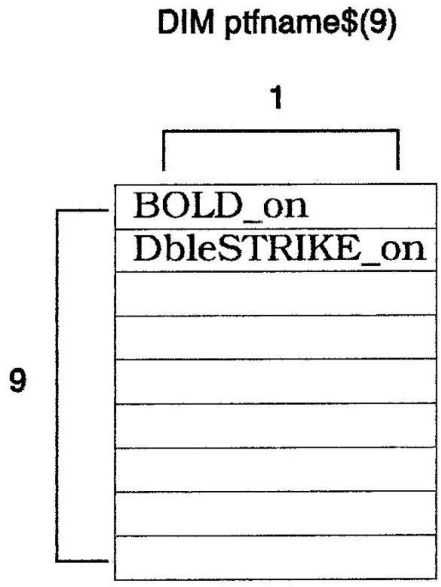


Figure 2b

To link the Epson control codes tabulated in the array ptfcode% with a command name or descriptor like Bold_on, required a new string array - DIM ptfname$(9), see figure 2b. The array can store up to 9 names. As long as *ptfname$(**1**)="Bold_on"* and *ptfcode%(1,27,69,0)* then the name and code are aligned. Unfortunately, SuperBASIC does not allow more than one type per array, e.g., *DIM ptface(9(ptfcode%(3),ptfname$))* is not allowed.

Pascal's RECORD structure does allow the above constructs. First of all you define the structure in Type definition and then assign this type to a variable name and finally, assign data to this variable in the main body of the listing. For example,

```
Type
        ptfacerec=RECORD
                ptfname_str:string[10];
                ptfcode:packed array[1..3] of integer;
        END;
Var
        ptface:ptfacerec;
BEGIN
        ptface[1].ptfname_str:="Bold_on";
        ptface[1].ptfcode[1]:=27;
        ptface[1].ptfcode[2]:=69;
        ptface[1].ptfcode[1]:=0;
END.
```

which equates to the following SuperBasic listing:-

```
10 DIM ptfname$(9)
20 DIM ptfcode%(9,3)
30 ptfname$="Bold_on"
40 ptfcode%(1,1)=27
50 ptfcode%(1,2)=69
60 ptfcode%(1,3)=0
70 STOP
```

Both versions are the same, except in Pascal one structure handles all the data, this does have its advantages.

In Pascal, the assignment of a array variable of type 'record' (i.e., *recordvar[index].field:=data;*) is almost identical to normal array variable assignment except without the field extension (i.e., *arrayvar[index]:=data;*). However, as in the previous example, typing out long assignments is extremely tedious. Fortunately, this can be circumvented using the Pascal WITH statement. This saves a lot of typing, especially when it is embedded at the start of a procedure or loop.

```
BEGIN
        With ptface[n] do
                begin {with}
                ........

                ptfname_str:="Bold_on";
                ptfcode[1]:=27;
                ptfcode[2]:=69;
```

```
            ptfcode[1]:=0;
            .........
END.
```

Refer to listing 4b, procedure - *read_data_ptf* or any of the recommend books for more information on the syntax and use of the WITH statement.

As discussed previously, Prospero's Pascal Compiler includes a QL specific window/graphic library providing a command set similar to that of SuperBASIC, see table 6, appendix 1. This library is only available if the programmer (you) declares it by either including the whole library (i.e., adding the line *{$I flp1_graphics_pas}* to the listing at the start, see listing 3) or declaring the appropriate commands using the EXTERNAL suffix as below:

```
PROCEDURE window(var output:text;width,height,xorg,yorg:integer);external;
```

I have used this technique for listing 4b, although using the $I include command would have saved a fair bit of typing. This disadvantage of the latter is the loss of meaningful syntax, type and ranges of the commands and the need to retain a copy of the reference manual. This is particularly important if you are updating the program or porting it to another system.

During compilation, ProPascal, adds the appropriate procedure and function names (e.g. PROCEDURE CLS(var screen:text;part:integer);external;) into the source text held in RAM, and declares them external to standard Pascal. The reason for external declaration, is to remind the compiler that these commands are not in the standard ProPascal library and that the binary extensions have to be retrieved from another library, i.e., graphics_pas. Libraries are quickly generated and are best grouped according to a particular theme or requirement e.g. statistics, QDOS, network, sound, etc.

The *init_win* procedure in both SuperBASIC and Pascal is almost the same. The key difference is that SuperBasic accesses the default input and output windows/consoles using numbers, i.e., *#0*, *#1*, *#2*, whereas, Pascal uses names, i.e., *input* and *output*. At machine code level both SuperBASIC and Pascal access these channels from QDOS, this will be covered in future chapters.

To send a control code from SuperBASIC to the printer in advance of text, e.g., to change the font, can be achieved with:-

```
Print ;
```

If you forget the semicolon (;) the printer head will move to the next line, as *PRINT*, sends a linefeed or is it a carriage return to the printer. *PRINT /* also moves the printer head to the next line. The Pascal equivalent for *PRINT ;* is *WRITE* and for *PRINT* or *PRINT /* its *WRITELN*.

Portability

The definition of portable source code could be written as "the source text of a program written in language X (i.e. Pascal or C) for computer A (i.e. Amiga), transferred to computer B (ie. PC compatible) and recompiled in language X without any modifications". This is 100% portability. However, this level of portability can only be achieved if you adhere strictly to the ISO Pascal or ANSI C standards. Neither of these standards account for Input/Output operations like GUI operations, file and graphics handling, etc. To this end, portability is restricted, generally 80% portability is achievable, depending on the number of I/O operations. Good Pascal and C programmers tend to isolate the I/O operations from the bulk of the source text, so that conversion to a different computer only requires the I/O part to be modified. Therefore a better description of portability would be "the isolation of computer specific Input/Output operations from the source text in order to minimise conversion rewriting" .

The Pascal listings used throughout this manual were compiled using Prospero ProPascal v1.51 for the QL and cross checked on the Amiga QL Emulator (ProPascal patch (see appendix 3) and ROM image were required). All programs were tested successfully on a QL with TRUMPCARD and on an Amiga A1200 and A600 emulating a QL. Throughout the chapters I have empahasised the need to consider portability and the advantages of including this in any program design, e.g., reusable code. I have taken a number of QL ProPascal programs and recompiled them successfully on an IBM PC running TurboPascal for windows, v1.5 with ease, for example typical changes include:

USES wincrt;	instead of	{$I flp1_graphics_pas}
CLRSCR;	instead of	cls(output,0);
LONGINT	instead of	INTEGER
BYTE	instead of	mcodetype (remove the line mcodetype=0..255;)
REAL*	instead of	LONGREAL
ELSE	instead of	OTHERWISE (part of the *case* construct)
COM	instead of	SER
LPT	instead of	PAR

* For Intel SX CPUs (without the FPU), only REAL is allowed. However, if you have a DX or Pentium then SINGLE, DOUBLE, EXTENDED, etc are available. Refer to TPW manual for details on ranges.

Matching up QL specific window/graphic commands with IBM's OS/2 or MS Windows GUI was easy due to the similarity. Changing some of the device names was easy, see listing 4b, procedure *prt_dev*. Appendix 4 includes a SuperBASIC to TurboPascal conversion program to speed up this process.

The next sections delve into assembly language/machine code, QDOS and how this interfaces with Pascal. Its not as daunting as it may sound because of the good links between ProPascal and QDOS and QDOS's simplicity.

LISTING 4a: Epson View Lite written for SuperBASIC (*note listing 4a differs from the main text above as it doesn't include ptfname$ but listing 4b does code for it*)

```
5 REMark EPSON VIEW Lite
10 pdev=1:REMark Ser1
15 ptf_maxtfaces=10: REMark includes endmarker
20 scrchan=1:prtchan=3:filechan=4
25 DIM ptfcode%(9,3)
30 ptf_msg$="Demo of Bold Printing":ptf_demo=1:REMark Bold on
35 init_win
40 init_epson_ptface 200: REMark load array with typeface data
45 prt_dev
50 send_prtmsg(ptf_demo,ptf_msg$)
55 REMark line 50 is the test line
60 REMark line 65 onwards copies a text file by line to the printer
65 INPUT(#scrchan,"Enter Typeface Number (e.g. 3 for NLQ on): ");sendtfn
70 get_filein
75 IF NOT EOF(#filechan)
80 THEN INPUT(#filechan,sendmsg$)
85 send_prtmsg(sendtfn,sendmsg$)
90 END IF
95 CLOSE #filechan:CLOSE #prtchan: STOP

100 DEFine PROCedure init_epson_ptface(linenum)
110 RESTORE linenum: REMark typeface data
120 REPeat loop
130 READ ptfcode%(ptfcnt,1)
140 IF ptfcode%(ptfcnt,1)=999 THEN EXIT loop
150 END IF
160 READ ptfcode%(ptfcnt,1),ptfcode%(ptfcnt,2),ptfcode%(ptfcnt,3)
170 ptfcnt=ptfcnt+1
180 END REPeat loop
190 END DEFine init_epson_ptface
195 REMark TYPEFACE CODES for EPSON PRINTERS
200 DATA 27,69,0 : REMark BOLD_on
210 DATA 27,71,0 : REMark DbleSTRIKE on
220 DATA 27,120,1 : REMark NLQ on
230 DATA 15,0,0 : REMark SMALL on
240 DATA 27,80,0: REMark PICA_on
250 DATA 27,70,0: REMark BOLD off
260 DATA 27,72,0: REMark DbleSTRIKE off
270 DATA 27,120,0 : REMark NLQ off
280 DATA 18,0,0: REMark SMALL off
290 DATA 999: REMark END_typeface list
```

```
300 DEFine PROCedure init_win
310 MODE 8
320 WINDOW(#scrchan,180,90,38,10)
330 INK(#scrchan,7)
340 PAPER(#scrchan,0)
350 BORDER(#scrchan,4,2)
360 CLS(#scrchan,0)
370 END DEFine init_win

400 DEFine PROCedure get_filein
410 INPUT(#scrchan,"Enter Filename (e.g. Flp1_demo_txt): ");qlfn$
420 OPEN#filechan,qlfn$
430 END DEFine get_filein

450 DEFine PROCedure prt_dev
460 SELect pdev
470 1=OPEN#prtchan,"ser1"
480 1=OPEN#prtchan,"ser2"
490 1=OPEN#prtchan,"par"
500 END SELect
510 END DEFine prt_dev

550 DEFine PROCedure send_prtmsg(sendtfn,sendmsg$)
560 IF sendtfn>0 and sendtfn<ptf_maxtfaces-1 THEN
570 PRINT#prtchan,chr$(ptfcode%(sendtfn,1),ptfcode%(sendtfn,2),ptfcode%(sendtfn,3)
580 PRINT#prtchan,sendmsg$\
590 END IF
600 END DEFine send_prtmsg
```

LISTING 4b: Epson View Lite written for Pascal

```
PROGRAM EPSON_VIEW_lite;

{version 1.01, A.F.Wilson, 3/12/93}
{copies an ASCII file to printer using specified Epson typeface)

{USES wincrt; for PC windows}

CONST
        def_pdev=1; {ser1} {LPT1 for PC windows}
        ptf_maxtfaces=10;
        ptf_msg='Demo of Bold Printing';
        ptf_demo='BOLD_on';
TYPE
        fileintype=text;
        lnumrange=1..99999;
         msgtype=string[255]; {ASCII line width max of 255}
        ptf_namtype=string[30];
        byte=0..255;
        ptf_tfrec=      RECORD
                                ptfname_str:ptf_namtype;
                                ptfcode:array[1..3] of byte;
                        END;
        ptf_tftype=array[1..ptf_maxtfaces] of ptf_tfrec;
VAR
        ptface:ptf_tftype;
        sendtfn_str:ptf_namtype;
        sendmsg_str:msgtype;
        printer:text;
        filein: fileintype;

{**** General Procedures used across platforms ****}

PROCEDURE init_epson_tfaces(var ptface:ptf_tftype);

CONST
        ptf_minlnum=200;
        ptf_steplnum=10;
TYPE
        ptf_range=0..ptf_maxtfaces; {must never be zero when indexing arrays}
VAR
        ptflinenum:lnumrange;
        ptfcnt:ptf_range;
```

```
PROCEDURE read_data_ptf(var ptface:ptf_tftype;ptfcnt:ptf_range;ptflinenum:lnumrange);
begin {read the data}
 with ptface[ptfcnt] do  {the WITH statement MUST come before the CASE}
  begin {with}
   case ptflinenum of
    190: begin {epson typeface control codes} end;
         200: begin ptfname_str:='BOLD_on';ptfcode[1]:=27;ptfcode[2]:=69;ptfcode[3]:=0;end;
         210: begin
         ptfname_str:='DbleSTRIKE_on';ptfcode[1]:=27;ptfcode[2]:=71;ptfcode[3]:=0;end;
         220: begin ptfname_str:='NLQ_on';ptfcode[1]:=27;ptfcode[2]:=120;ptfcode[3]:=0;end;
         230: begin ptfname_str:='SMALL_on';ptfcode[1]:=15;ptfcode[2]:=0;ptfcode[3]:=0;end;
         240: begin ptfname_str:='PICA_on';ptfcode[1]:=27;ptfcode[2]:=80;ptfcode[3]:=0;end;
         250: begin ptfname_str:='BOLD_off';ptfcode[1]:=27;ptfcode[2]:=70;ptfcode[3]:=0;end;
         260: begin
         ptfname_str:='DbleSTRIKE_off';ptfcode[1]:=27;ptfcode[2]:=72;ptfcode[3]:=0;end;
         270: begin ptfname_str:='NLQ_off';ptfcode[1]:=27;ptfcode[2]:=120;ptfcode[3]:=0;end;
         280: begin ptfname_str:='SMALL_off';ptfcode[1]:=27;ptfcode[2]:=0;ptfcode[3]:=0;end;
         290: ptfname_str:='END_tface';
  end; {case}
 end; {with}
end; {ptface data}

begin {init epson typefaces}
         ptflinenum:=ptf_minlnum-ptf_steplnum;
         ptfcnt:=0;
   Repeat
                  ptfcnt:=ptfcnt+1;
                  ptflinenum:=ptflinenum+ptf_steplnum;
                  read_data_ptf(ptface,ptfcnt,ptflinenum);
   until (ptface[ptfcnt].ptfname_str='END_tface');
end;

PROCEDURE send_prtmsg(var printer:text;var ptface:ptf_tftype;
                                                sendtfn_str:ptf_namtype;sendmsg_str:msgtype);
VAR
   scnt:1..ptf_maxtfaces;
   find_flag:boolean;
Begin
  find_flag:=false;
  scnt:=1;
  While (find_flag=false) or (ptface[scnt].ptfname_str='END_tface') do
    begin
          with ptface[scnt] do
      begin {with}
                  if ptfname_str=sendtfn_str
```

```
        then begin
                write(printer,chr(ptfcode[1]),chr(ptfcode[2]),chr(ptfcode[3]));
                writeln(printer,sendmsg_str);
         find_flag:=true;
        end {if}
       else scnt:=scnt+1;
     end; {with}
   end; {while}
end; {proc}


PROCEDURE get_filein(var filein:fileintype);
VAR
        qlfn_str:string[36];  {can be used for PC}
begin
        write(output,'enter filename (e.g., flp1_demo_txt): ');
                                {e.g., a:\demo.txt for PCs}
        readln(input,qlfn_str);
        assign(filein,qlfn_str);
   reset(filein);  {open file, ready for reading}
end;

PROCEDURE get_ptfname(var sendtfn_str:ptf_namtype);
begin
        write(output,'enter typeface name (e.g., DbleSTRIKE_on) ');
        readln(input,sendtfn_str);
end;

{**** QL Specific Graphic & Device Procedures ****}

PROCEDURE mode(highres:boolean);external;
PROCEDURE window(var output:text;width,height,xorg,yorg:integer);external;
PROCEDURE ink(var output:text;colour:integer);external;
PROCEDURE paper(var output:text;colour:integer);external;
PROCEDURE border(var output:text;width,colour:integer);external;
PROCEDURE cls(var output:text;part:integer);external;

PROCEDURE init_win;
begin
        mode(false); {mode 8 ; true=mode 4}
        window(output,180,90,38,40); {redefine main window}
        ink(output,7);      {white}
        paper(output,0);    {black}
        border(output,4,2);  {red}
   cls(output,0);                   {clrscr for PC windows}
```

```
end;


PROCEDURE prt_dev(var printer:text;pdev:integer);
Begin
case pdev of
        1: assign(printer,'SER1');   {COM1 for PC windows}
        2: assign(printer,'SER2');   {COM2 for PC windows}
        3: assign(printer,'PAR');    {LPT for PC windows}
end; {case}
rewrite(printer);
End;


BEGIN {main}

        init_win;
        prt_dev(printer,def_pdev);
   init_epson_tfaces(ptface);
   send_prtmsg(printer,ptface,ptf_demo,ptf_msg);
        {END.     to test, then..}

        get_filein(filein);       {part of the menu system}
        get_ptfname(sendtfn_str); {part of the new menu system}
   while not eof(filein) do
```

4. Interfacing with QDOS

QDOS is a well balanced operating system offering tremendous power and flexibility without compromising speed. I only wish Microsoft's Windows was as delightful to program. Prospero's ProPascal interfaces easily with QDOS. The ProPascal command set includes a limited gateway to QDOS via the QTRAP command. The QTRAP syntax and usage is adequately documented in the manuals and examples supplied by Prospero. The ProPascal Manual also covers interfacing and linking directly with assembly language or machine code and should be read inconjunction with this text. Three others texts worth reading are: "The Sinclair QDOS Companion" by Andrew Pennell; "Advanced QL Machine Code" by Adam Denning and "Systematic Machine Code Programming", SQLW series of articles. Two development products worth investing in are: GST QL Macro Assembler and the QDOS Graphical/Pointer extensions - QPTR disc and Manual from Jochen Merz Software. I also highly commend the QPAC2 WIMP environment (see appendix 3).

I don't intend covering the Motorola 68000 assembly language in any depth, this as stated above is adequately covered elsewhere, therefore I am assuming a basic knowledge. I will instead review and discuss the key learning points used to improve both the cursor and file capability of ProPascal. It should be possible to extended the skills and processes used throughout the remaining chapters to utilise the vast library of assembly language listings from magazines and directly link them with Pascal. However, for obvious reasons, the use of QL Specific QDOS calls will hinder any conversion to other systems, with the proviso, that hopefully many of these routines are already incorporated into the new operating systems like WARP and Windows NT.

Interfacing either SuperBASIC or Pascal with QDOS requires that certain criteria are met. For instance, certain assembly language CPU registers must be left unaltered on returning from a linked assembly language module, refer to the manual. In addition, the QDOS gateways called TRAPS also impose restrictions on status of the CPU registers. Andrew Pennell's book is a must if you are serious about using the QDOS Traps (subroutines). e.g., to switch the cursor on use the QDOS subroutine SD.CURE (set TRAP to #3 and register D0 with $0E). Note that, if D0=$0F then the cursor is switched off (SD.CURS). The book then goes onto specify :-

1. the **entry** conditions (e.g., A0.L contains the Channel ID)
2. the **exit** conditions (e.g., D1 corrupted)
3. the **errors** that can occur (e.g., -4 out of range if cursor will not fit in window).
4. Finally the **actions** describes the command detail and any parameters passed, e.g., to redefine a window, set TRAP #3, D0=$0D. The parameters that need to be passed to QDOS are the width, depth, x-pos and y-pos.

This should be clear as mud!. I hope the recommended texts are used inconjunction with this text.

Now down to an actual example. One of the drawbacks of the QL SuperBASIC language was the poor cursor control, e.g., to avoid the numerous blank or null input statements to ensure the cursor appears at the correct position on the screen or console. Tony Tebby's Toolkit 2,

which is standard to all disc interfaces, addressed this problem with extension cursen and curdis, refer to table 6, appendix 1. In actual fact, the DIY solution to this problem is very simple and therefore an ideal example to kick off this look at interfacing Pascal with QDOS.

Listing 5 (module cursorql) should be typed into your editor and saved to disc as *flp2_cursorql_asm*. Now compile this using your assembler, e.g., *exec flp1_mac*. At the GST QL macro assembler prompt type:

flp2_cursorql -errors -bin flp1_cursorql_rel

This line loads the *cursorql_asm* file and assembles it into executable code format (e.g., -bin), reporting any errors (e.g, -errors). The second statement then converts the machine code into a Sinclair REL or relocatable code file. The _REL file ensures that the code is not based on absolute RAM addresses but allows the code to load anywhere in RAM, ensure no clashes with other programs trying to use the same address space or RAM. To interface this code with Pascal requires the ability to link _REL files. All the Pascal libraries and user programs are in the _REL format. To generate a executable binary file requires that programs and the libraries it uses are linked together to give the finished product. The utility *pas_link* performs this duty, more on this later.

Pascal listing 6 (program testcursorql) should be typed into your editor and saved to disc as *flp2_testcursorql_pas*. Now convert this into a _REL program using the propas compiler. Do NOT link at this stage. This must be saved to the same disc as cursorql_rel. We now have *cursor_rel* and *testcursorql_rel* on flp2_. Before we link the two files to give the test program, we have to understand how the process of interfacing Pascal with assembly language.

Firstly, lets review the cursorql module (listing 5). The first line tells the assembler that we are writing a module or subroutine called cursorql. Within this module, we are going to define (xdef is the assembler command) two procedures *curson* (cursor on) and *cursoff* (cursor off). The *code* to do this comes next. First we list the constants, which in this case are the two Trap #3 codes for switching the cursor on ($0E) and off ($0F). Next comes the cursor on subroutine, *curson*.

As stated earlier, Pascal makes demands of the assembler programmer to ensure both the stack (workspace) and a0 register (holds the Pascal return address) are left unaltered. If this did not happen, you would scramble the Pascal code, assuming you were able to get back. In both routines, *move.l 4(sp),a0* ensures the Pascal return address is saved and *move.l (sp)+,a0* restores it and *jmp (a0)* returns control back to the Pascal calling program, i.e. **testcursorql.** To reset the stack use *adda.w #4,sp***.** The code between these statements sets and resets the cursor.

curson	**cursoff**	**Description**
moveq #sd_cure,d0	moveq #sd_curs,d0	cursor on/off
moveq #-1,d3	moveq #-1,d3	cursor to flash as required indefinitely
trap #3	trap #3	selects on or off when d0 passed to trap #3
moveq #0,d0	moveq #0,d0	clear error codes as not important.

Lastly, the END instruction signifies the end of the code to the compiler. I think you'll agree that QDOS is very accessible and powerful, you may even say what's all the fuss about!.

Before, moving onto the main Pascal code, it is worth comparing the Pascal interface protocol with that of SuperBASIC. SuperBASIC, like Pascal needs to keep track of the stack and needs to know the return address. this is hidden away in the QL ROM and accessed via an indirect addressing vector $110 (BP.INIT). Refer to the two QL book references for a fuller description. SuperBASIC also needs to know the number and names of any procedures and/or functions. SuperBASIC needs the following information:

Code		**Description**
move.w $110,a0		access vector BP.INIT
lea procs(PC),a1		the SuperBASIC defn is relative to here
JSR (a0)		call vector, register a1 printing to PROCS
RTS		return to SuperBASIC
PROCS:	dc.w 2	Number of procedures defined
	dc.w curson-*	where the 1st procedure is located
	dc.b 6	length of 1st procedure name
	dc.b 'curson'	1st procedure name
	dc.w cursoff-*	where the 2nd procedure is located
	dc.b 7	length of 2nd procedure name
	dc.b 'cursoff'	2nd procedure name
	dc.w 0	end of procedure list
	dc.w 0	no functions defined
	dc.w 0	end of function list
CURSON:		start of 1st procedure

The SuperBASIC interpreter now adds *curson* and *cursoff* to the list of procedures and the programmer can now use these procedures. this differs from Pascal which first has to define the procedure with parameters/variables as external to the Pascal language, this informs the compiler that this code is located in a new library or _rel file, e.g.,

PROCEDURE curson(chanio:integer);external;

The next step in the Pascal process is to link the externally defined procedures with the Pascal libraries to get the end product. With the SuperBASIC interpreter this is implicit. In Pascal, the two _rel files are linked using the linker utility:

exec flp1_link (GST linker) or *exec flp1_pas_link* (ProPascal linker)

Once loaded, type

flp2_testcursorql flp2_cursorql flp1_pas

This command links the Pascal libraries with the cursorql and testcursorql _rel files. You should end up with a program called testcursorql_bin. To run this program use

exec flp2_testcursorql_bin

What I prefer to do is to load the pas_link file on flp1_ into my editor and add all my new library _rel's like cursorql, formatql, dirql etc to the list and then re-save. This means the linking process becomes

flp2_testcursorql flp1_pas

The advantage of this is that it hides the libraries, reduces typing and can at a letter date be merged to generate a new library, e.g., Statistics or extended file handling, etc.

The Pascal program testcursorql, does what it says. Rather than waste anymore time, try and see. The only point worth noting, is that in SuperBASIC, #1 is used to access the windows, whereas in Pascal names like input or output are used. *winio* is a new console, which Pascal assigns a low level channel number, e.g., $0000000 is #0 and $00010001 is #1. This is exactly the same as happens in SuperBASIC. The channel number is made up of two 16-bit words, the tag and the chan. It is this channel number that QDOS uses to track the windows on the screen. for example, when a window (can be a screen or console) is opened, QDOS assigns it a tag and a chan which make up the channel ID. If, SuperBASIC opened window #2, QDOS would assign this a tag of 0002 and a chan of 0002 which gives a channel ID of $00020002. However, say, this windows is reopened, QDOS will keep the same chan value but the tag is increased by one, to 0003, giving a new channel ID of $00030002. For a better understanding, refer to the previously referenced books and articles.

LISTING 5: module cursorql

```
module cursorql
 xdef curson              ;Pascal procedure name
 xdef cursoff             ;Pascal procedure name
 section .code

sd_cure  EQU $0E
sd_curs  EQU $0F

curson:     move.l 4(sp),a0      ;get channel ID (ie,input/output/?) from stack
            moveq #sd_cure,d0    ;cursor on
            moveq #-1,d3         ;timeout=infinite
            trap #3              ;call QDOS
            moveq #0,d0          ;clear error code held in d0. Possible errors
                                 ;are either 'Bad Parameter' or 'Channel not open'
            move.l (sp)+,a0      ;get Pascal return address
            adda.w #4,sp         ;reset stack
            jmp (a0)             ;back to pascal

cursoff:    move.l 4(sp),a0      ;get channel ID (ie,input/output/etc) from stack
            moveq #sd_curs,d0    ;cursor off
            moveq #-1,d3         ;timeout = infinite
            trap #3              ;call QDOS
            moveq #0,d0          ;clear error code held in d0. Possible errors
                                 ;are either 'Bad Parameter' or 'Channel not open'
            move.l (sp)+,a0      ;get Pascal return address
            adda.w #4,sp         ;reset stack
            jmp (a0)             ;back to Pascal

 END
```

LISTING 6: testing cursor QL module from within Pascal.

```
PROGRAM testcursorql(input,output);

{A.F.Wilson 8/9/92, version 1.0  }
{lines=  68  code= 856  data=  25}

VAR
  ch:char;
  chaninp,chanout,chanio:integer;
  winio:text;
  textstr:array [1..6] of char;
  cnt:0..9;

FUNCTION consilent:char;external;

PROCEDURE curson(chanio:integer);external;

PROCEDURE cursoff(chanio:integer);external;

PROCEDURE cls(var winio:text;part:integer);external;

BEGIN
  {open a window for input/output - QL device CON}
  assign(winio,'con_448x32a32x180');
  rewrite(winio);
  cls(output,0);
  cls(winio,0);

  {get the channel ID for all defined devices}
  chaninp:=handle(input);   {default input only}
  chanout:=handle(output);  {default output only,ie SCR type}
  chanio:=handle(winio);    {new input/output device,ie CON type}

  {write the tag numbers to the output window for reference}
  writeln(output,'channel ID for INPUT ',chaninp,' for OUTPUT '
         ,chanout,' for winio ',chanio);

  {normal Pascal input/output structure}
  writeln(output);
  write(output,'Input text "demo 1" ');
  repeat
    read(input,ch);
    if ch IN ['a'..'z','0'..'9',' '] then write(output,ch);
  until eoln;

 {Consilent without cursor extensions}
 write(winio,'Input text "demo 2" ');
 repeat
```

```
   ch:=consilent;
   if ch IN ['a'..'z','0'..'9',' '] then write(winio,ch);
until ch=chr(10);   {EOLn doesn't work here}
writeln(winio);     {send EOLn at line end}

{Consilent with cursor extensions}
write(winio,'Input text "demo 3" ');
cursoff(chaninp);
curson(chanio);
repeat
   ch:=consilent;
   if ch IN ['a'..'z','0'..'9',' '] then write(winio,ch);
until ch=chr(10);
writeln(winio); {send EOLn at line end}

{reset cursor}
cursoff(chanio);
curson(chaninp);


END.
```

5. Disc Formatting

This last chapter demonstrated the 6 processes involved in writing Pascal/assembly language programs:

1. Write the Pascal source file (e.g., testcursorql_pas).
2. Generate the relocateable binary file (e.g.,testcursorql _rel) from the Pascal source file using the Propas compiler.
3. Write the assembly language modules (e.g., cursorql_asm)
4. Assembly or compile the modules into a relocatable binary file (e.g., cursorql_rel).
5. Edit the pas_link file to include the cursorql_rel file in the list of libraries - for ease of use.
6. Generate the executable binary file (e.g., testcursorql_bin). This is done by linking the testcursor_rel file with pas_link (containing the library list).

Do not constrain your creativity when writing Pascal procedures by wondering about QDOS. If you have to write assembly language modules to interface with QDOS, then do so when you have defined your program requirements. Only then, review and consider the implications with respect to assembly language/QDOS, this should not pose a problem as this is integral in the development phase of writing any program. It is always better modify than start from scratch.

The big surprise with the Prospero's ProPascal implementation on the QL was the lack of format and directory file commands, considering the quality and breadth of graphics and other file extensions. My first thought was ROM or time constraints, however, no such extensions were found in subsequent releases, I was using version 1.51. These two commands are the linchpins of any application and as such, probably hindered the effective use of this excellent language to generate programs. The only commercial program I am aware of that used ProPascal was TurboQuill+. The examples in this and subsequent chapter address these shortfalls.

Listing 7 (module formatql), uses the QDOS io_format trap. As with the QL SuperBASIC command:

format flp1_backup

The QDOS io_format trap requires that both the device name (dev_name, e.g., flp1_) and the device identification name (dev_idname, e.g., backup) are passed as parameters. The limitations or syntax of both parameters are the same for the QL format command, see table below:

dev_len	dc.w $0F	Length of dev_name (5) + dev_idname (10) = 15 characters
dev_name	dc.b 'mdv1_'	default set to mdv1_
dev_idname	dc.b ' ',0	default set to 10 spaces. The 0 ensures that the data totals 16 bytes, i.e., ends the code on an even word boundry in RAM.

The formatql module receives the dev_name and dev_idname from the Pascal stack. The stack information is copied over the dev_name/idname defaults and used by the

```
moveq #i0_format,d0
trap #2
```

This will tell QDOS to format the specified physical media, e.g., microdrive (211 sectors) or floppy drive (1440 sectors). If the format fails because of bad medium or whatever, QDOS returns the error code in d0, e.g., -14 for format failure. On the other hand, a successful format will return total sectors formatted (e.g. totsec) and how many of these are good sectors (e.g. goodsec). The error code, total and good sectors are returned in d0, d2 and d1 respectively and then stored on the stack to be copied into the relevant Pascal variables errcode, goodsec and totsec.

The assembly language listing is again concise and readily understandable. This again emphasises the elegance of both the QDOS ROM and the Motorola CPU assembly language. I am positive that if the original IBM PC had adopted the Motorola CPU series in preference to the Intel 80x86 series that superior multitasking operating systems would be common place and not just making an appearance late in 1995.

The Pascal procedures to relay this format information is just as elegant. The three procedures, possibly four (see later) that are needed to ensure the success of the formatting process are:

getqlfn	- get the media name and the filename/format id name.
formatql	- pass getqlfn information to the formatql module.
reporterr	- report the unintelligible error code as understandable text.

The main code utilises the totsec and goodsec to report the success of the formatting process and also include the number of bad sectors (badsec=totsec-goodsec). The main code demonstrates the usage/syntax of the new commands and should be procedurised thus simplifying its use further.

Both the procedures reporterr and getqlfn are durable and reusable. QDOS only has 22 feeble error codes and reporterr translate all of them into the appropriate meaningful text. The getqlfn procedure ensures full filename/format name syntax checking. It is worth pointing out that the QDOS format trap does not include ram disc drives. The extension to include ram discs is built into the disc drive interface. Unfortunately, I'm not sure how this is accessed as yet. Hard drives and sub-directories also fall into this category.

LISTING 7: module formatql

```
module formatql
xdef formatql
section .code

 io_format:   equ 3                ;trap 2
 dev_len:     dc.w $0f             ;5 bytes for dev_name and 10 for id_name
 dev_name:    dc.b 'mdv1_'         ;mdv and flp only not RAM
 dev_idname: dc.b '         ',0    ;align on even word boundry

formatql:       lea dev_len(pc),a0       ;copy dev_name and dev_idname from
                movea.l 4(sp),a1         ;Pascal and store them in the
                move.l (a1),2(a0)        ;above lacations.
                move.l 4(a1),6(a0)
                move.l 8(a1),10(a0)
                move.l 12(a1),14(a0)
                moveq #io_format,d0      ;format Ql disc
                trap #2
                movea.l 8(sp),a1         ;a1=Pascal address where errcode is stored
                move.b d0,(a1)           ;if error occurs, errcode=error returned in d0
                movea.l 12(sp),a1        ;a1=Pascal address where totsec is stored
                move.w d2,(a1)           ;totsec=value held in d2
                movea.l 16(sp),a1        ;a1=Pascal address where goodsec is stored
                move.w d1,(a1)           ;goodsec=value held in d1
                move.l (sp)+,a0          ;a0=Pascal link return address
                adda.w #16,sp            ;remove 4 variables from stack,ie reset it
                jmp (a0)                 ;return to pascal.

 END
```

LISTING 8: test formatql module from Pascal

```
program testformatql(input,output);

{lines= 191  Code=2948  Data= 36}
{A.F.Wilson ,12/3/92 ,version 1.0}
{only for flp and mdv but not RAM}

type

sectype=0..65535;
errtype=-21..0;
qlfntype=packed array[1..36] of char;

var

goodsec,totsec,badsec:sectype;
idevice,odevice:text;
formatname,qlfname:qlfntype;
ch:char;
fnopt:boolean;
errcode:errtype;

procedure formatql(var goodsec:sectype;var totsec:sectype;var
errcode:errtype;formatname:qlfntype);external;


Procedure getqlfn(var idevice,odevice:text; var qlfname:qlfntype;fnopt:boolean);

{this procedure can use STRING or ARRAY filename variables }
{ CR with no text constituents a blank filename and exits  }

const
blankfn='                                    ';
qlfnlen=36;
qlfmtlen=15;
promptfn=' Enter Ql Media and Filename: ';
promptfmt='Enter Ql Format Media and Discname : ';

var
fnfmtlen,cnt:0..36;
ch:char;
fail:0..3;

begin
Repeat
fail:=0;
qlfname:=blankfn;
```

```
if fnopt=true then
 begin
   write(odevice,promptfn);
   fnfmtlen:=qlfnlen;
 end
else
  begin
    write(odevice,promptfmt);
    fnfmtlen:=qlfmtlen;
end; {if}

cnt:=0;
repeat
read(idevice,ch);
if (cnt<fnfmtlen) and ((((ch>='A') and (ch<='Z')) or ((ch>='a') and (ch<='z'))) or (((ch>='0')
and (ch<='9')) or (ch='_'))) then
  begin
  cnt:=cnt+1;
  qlfname[cnt]:=ch;
  end;
until (cnt>fnfmtlen) or Eoln;
readln(idevice);

if cnt>5 then
 begin

 if qlfname[5]<>'_' then fail:=1
 else
   begin
    ch:=qlfname[1];
    case ch of
       'F','f':if (((qlfname[2]<>'L') and (qlfname[2]<>'l')) or ((qlfname[3]<>'P') and
(qlfname[3]<>'p'))) or ((qlfname[4]<'1') or (qlfname[4]>'4')) then fail:=1;
       'M','m':if (((qlfname[2]<>'D') and (qlfname[2]<>'d')) or ((qlfname[3]<>'V') and
(qlfname[3]<>'v'))) or ((qlfname[4]<'1') or (qlfname[4]>'4')) then fail:=1;
       'R','r':if (((qlfname[2]<>'A') and (qlfname[2]<>'a')) or ((qlfname[3]<>'M') and
(qlfname[3]<>'m'))) or ((qlfname[4]<'1') or (qlfname[4]>'4')) then fail:=1;
     otherwise fail:=1;
    end; {case}



{if ((((ch<>'F') and (ch<>'f')) and ((ch<>'M') and (ch<>'m'))) and ((ch<>'R') and (ch<>'r')))
and (fail<>0) then fail:=1}
{ the above line is not needed due to the otherwise clause}

 end; {if}
end; {if}
```

```
case fail of
0: if cnt<6 then qlfname:=blankfn; { else valid specification }
1: writeln(odevice,'Media specification wrong ( flp,mdv,ram 1->4 only ) ');
{2: writeln(odevice,'Invalid characters in filename ( A..Z , a..z , 0..9 and _ only ) ');}
{ option 2 taken care of at the input stage}
end; {case}


until fail=0;

end;

procedure reporterr(var idevice,odevice:text;errcode:errtype);

const
  err_nc='ERROR: routine not complete';
  err_nj='ERROR: invalid job or job does not exist';
  err_om='ERROR: out of memory';
  err_or='ERROR: out of range';
  err_bo='ERROR: buffer overflow';
  err_no='ERROR: not open';
  err_nf='ERROR: not found';
  err_ex='ERROR: file already exists';
  err_iu='ERROR: file/device already in use';
  err_ef='ERROR: end of file (EOF)';
  err_df='ERROR: drive full';
  err_bn='ERROR: bad name';
  err_te='ERROR: transmission error on RS232';
  err_ff='ERROR: format failed';
  err_bp='ERROR: bad parameter';
  err_fe='ERROR: file error';
  err_xp='ERROR: expression error';
  err_ov='ERROR: overflow ,ie when division by ZERO';
  err_ni='ERROR: not implemeneted';
  err_ro='ERROR: read only';
  err_bl='ERROR: bad line';

begin
  case errcode of
  -1: writeln(odevice,err_nc);
  -2: writeln(odevice,err_nj);
  -3: writeln(odevice,err_om);
  -4: writeln(odevice,err_or);
  -5: writeln(odevice,err_bo);
  -6: writeln(odevice,err_no);
  -7: writeln(odevice,err_nf);
  -8: writeln(odevice,err_ex);
  -9: writeln(odevice,err_iu);
 -10: writeln(odevice,err_ef);
```

```
 -11: writeln(odevice,err_df);
 -12: writeln(odevice,err_bn);
 -13: writeln(odevice,err_te);
 -14: writeln(odevice,err_ff);
 -15: writeln(odevice,err_bp);
 -16: writeln(odevice,err_fe);
 -17: writeln(odevice,err_xp);
 -18: writeln(odevice,err_ov);
 -19: writeln(odevice,err_ni);
 -20: writeln(odevice,err_ro);
 -21: writeln(odevice,err_bl);
   0: {okay}
 end; {case}

writeln(odevice);
writeln(odevice,'Press ENTER to continue ');
readln;

end; { error proc }

BEGIN {main}
   fnopt:=false;
   getqlfn(input,output,qlfname,fnopt);
   goodsec:=0;
   totsec:=0;
   errcode:=0;
   if qlfname[1]<>' ' then
    begin
     formatql(goodsec,totsec,errcode,qlfname);
     if errcode<0 then reporterr(input,output,errcode)
     else
       begin
         badsec:=totsec-goodsec;
         writeln(output,'total sectors = ',totsec);
         writeln(output,'good sectors  = ',goodsec);
         writeln(output,'bad sectors   = ',badsec);
         writeln(output);
         writeln(output,' format finished, press ENTER to exit ');
         readln;
    end; {if}
  end; {if}
END.
```

6. Directory Enquiries

Passing variables and/or parameters from Pascal to/from assembly language is achieved by either adding or subtracting data temporarily stored on the variable stack. The Pascal return address is also passed to the stack whenever an external module is called. Without the return address, the program will crash. A similar crash will result if the stack is corrupted in anyway. Whenever information is placed onto the stack, the stack decreases by either 1,2 or 4 bytes depending on the data passed. However, odd data must be paired to avoid mishaps with the program counter which requires data to be stored on “word boundaries or at even addresses”. QDOS uses a TOP to BOTTOM stack system sometimes referred to as first in last out (FILO). This means when we return information we actually add to the stack. This explains the addw.w #?,sp prior to returning to Pascal with jmp (a0).

This is the last chapter in this Pascal series, as expected I have left the best and most difficult to last, the dirql module (listing 9) uses 5 different QDOS traps. The first thing to notice about the dirql module is the structured and clear layout of the code. This is essential when de-bugging, improving or just understanding the code.

The structure of the QL file system is simple and until recently superior to that offered by IBM PC Compatibles running DOS or Windows. Whether the media is microdrive, floppy drive or ramdisc, a section of the disc is reserved for holding key information pertaining to the size, location and name of the file(s) during formatting. Without this information, QDOS would find it difficult to locate information, unless stored in a sequential manner as used by music cassettes or video tapes. The latter is obviously slow and doesn't offer fast random access.

The QL directory information is contained in a reserved area. This reserved area can at present hold upto 96 entries. These entries contain the directory header which is 64 bytes long. The directory header has the following format:

Offset from Start	Data Size	Description	Default
0	Long Word (4)	File length	
4	Byte (1)	File access	0
5	Byte (1)	File type	0
6-13	Bytes (1x8)	File type dependent info.	
14-15	Word (2)	File name length	
16-51	Bytes (1x 36)	File name in characters	
52-63		ignore	

Whenever, a new file is saved, the above information is copied to the end of the directory entries list and the file data is saved to disc. The reverse operation, e.g., deletion of a file from a disc does not overwrite any data but simply overwrites both the file length and file name length by putting 0 in the appropriate entry location. Data recovery or undelete utilities rely on this fact when recovering those lost files. It also means that simple deletion is no sure way of removing confidential information from floppy or microdiscs. So unless your media has been re-formatted where a single value is written across the whole disc, e.g., $e5, do not throw or give away your discs if your data is confidential.

Before moving onto the actual code, the file length is the length of the data plus 64 for the header, so if you want to know the actual file length as displayed by SuperBASIC, subtract 64. Personally, I prefer to know the total file length and not just the data length.

The procedure created in Pascal to action a directory read and to store the directory information is:

Dirql(dirlist,errcode,medianame)

Errorcode, as exactly the same as that described in the last chapter. The medianame informs QDOS whether its microdrive (mdv**?**_), floppy drive (flp**?**_) or ram disc (ram**?**_) to be read. The dirlist is a predefined array of 96 entries of 64 byte (6144 bytes). For simplicities sake, I have used a continuous array space of 6144 bytes as opposed to a 2 dimensional array [1..96.1..64]. **Please note, if the array is under-sized, the directory information overwrites other variables or code and results in a system carsh. No problems if oversized. Update the size of the array as required.**

The assembly language module then performs the directory operation, more on this later. I have used the ProPascal function addr() to report the RAM location of the directory list. This means I can use a hexdump program to interrogate the variable storage area for additional information, if required. I use it as a check that the variable dirlist points to the same location before and after the assembly language operation. The rest of the Pascal code, displays the directory entries until end of file (EOF) is found. This directory could be displayed in whatever format the programmer requires, in this example, a simple scrolling display suffices.

The dirql module, was designed in this case according to particular functions:

time_out	System variables
dirql	Get Pascal parameters
dir_open	Select and open the directory
dir_title	Get number of sectors and medium name
dir_readhdr	get the directory information
dir_emptyhdr	checks for filelength or filename lengths equal to zero or EOF
dir_error	save the error code if generated
dir_close	close the channel
dir_quit	return to Pascal

Each of the above sub-sections is adequately described in listing 9, however, for those who would like a little more detail should read on a bit.

moveq #time_out,d1	set to indefinitely
moveq #open_dir,d3	selects to open a directory
moveq #io_open,d0	open the directory
trap #2	this trap tells QDOS to select and then open a directory
tst.l d0 bne.s dir_error	If an error occurs, due to missing media (not found) or ... then the error code is stored in register d0. If d0 is not equal to zero as a result of an error then the program branches to dir_quit (not dir_error - read on).
movea.l a0,a2	QDOS generates a channel ID to identify this particular directory operation. This ID is similar to the ID used for selecting the window cursor. The ID is unique to this drive (e.g., flp1_) versus another (e.g., flp2_). This ID must be saved until the end as this ID is needed to close the channel on completion of task.

The reason, the program at this stage branches to dir_quit because no channel ID has been generated and we already have the error code in d0, therefore, we don't need to recover d0 or close any channels, therefore, dir_error/dir_close are redundant in this instance.

I hope the above short explanation clarifies any unanswered questions, from here on in, your on your own. EXPERIMENTATION is one of the best ways to learn.

LISTING 9: module dirql

```
 module dirql
   xdef dirql
   section .code

time_out      equ -1
io_open       equ 1       ; trap 2
io_close      equ 2       ; trap 2
open_dir      equ 4       ; trap 2
fs_mdinf      equ $45     ; trap 3
io_fstrg      equ $3      ; trap 3
fs_hdrlen      equ $40
dev_len       dc.w 5
dev_name      dc.b 'flp1_',0
med_name      dc.b $0a

 ;d0 returns the error code,<0 then error; =0 then no error
 ;d4-d7 and a4-a7 are never corrupted
 ;a3-a6 must be unchanged as required by Pascal
 ;on return the link and all stack params must have been removed
 ;remember stack is organised top(start) to bottom(end), therefore
 ;when removing parameters from the stack, use ADD and not SUB.

dirql       lea dev_len(pc),a0
            movea.l 4(sp),a2            ;get dev_name from stack
            move.l (a2),2(a0)           ;copy device to dev_name
            move.l 12(sp),d5            ;get address of array

dir_open  moveq #time_out,d1
            moveq #open_dir,d3          ;select directory
            moveq #io_open,d0           ;open directory
            trap #2
            tst.l d0                    ;d0 returns the error code
            bne.s dir_quit
            movea.l a0,a2               ;save dir ID, needed for closing

dir_title   moveq  #fs_mdinf,d0
            lea med_name,a1
            movea.l a2,a0
            moveq  #time_out,d3         ;read medium information
            trap #3
            tst.l d0                    ;d0 returns the error code
            bne.s dir_error
            move.l d1,d4                ;d4=empty/good sectors
                                        ;med_name=medium name
                                        ;these could be linked to Pascal if needed
dir_readhdr     moveq #io_fstrg,d0
                moveq #fs_hdrlen,d2
```

```
                movea.l a2,a0           ;a0 points to new ID
                movea.l d5,a1           ;pointer address from Pascal
                moveq #time_out,d3      ;file header read
                trap #3                 ;d1.w returns number of bytes read
                tst.l d0                ;d0 returns the error code
                bne.s dir_error         ;end of file (EOF)

dir_emptyhdr    movea.l d5,a1
                tst.l (a1)              ;check if filelength=0
                beq.s dir_readhdr
                tst.w 14(a1)            ;check if filenamelength=0
                beq.s dir_readhdr
                addi.l #fs_hdrlen,d5    ;move d5 to next entry ram location
                bra dir_readhdr         ;repeat until EOF

dir_error       move.l d0,d4            ;save error code.
dir_close       movea.l a2,a0           ;restore dir ID
                moveq #io_close,d0      ;close dir ID
                trap #2
                move.l d4,d0            ;restore error code.
dir_quit        movea.l 8(sp),a2
                move.b d0,(a2)          ;save error code
                move.l (sp)+,a0         ;Pascal link address
                adda.w #12,sp           ;remove parameters from stack
                jmp (a0)                ;return to Pascal

 END
```

LISTING 10: test dirql module from Pascal

```
program testdirql(input,output);

{A.F.Wilson 20/8/92, version 1.0}
{lines=  139   code= 2252    data= 6180 }

{filenames=testdirql_pas & dirql_asm gives testdirql_rel & dirql_rel}
{the REL files are linked  together to give testdirql_bin}

CONST

dirhdrlen=64;
direntries=96;  {dirlist array size=64*96}
SP=32;
LF=13;
CR=10;

TYPE

sectype=0..65535;
errtype=-21..0;
mediatype=packed array[1..5] of char;
dirtype=packed array[1..6144] of char; {system will crash if array to small}

VAR

idevice,odevice:text;
medianame:mediatype;
ch:char;
dirlist:dirtype;
ramaddr,cnt,cnt1:integer;
errcode:errtype;

procedure dirql(var dirlist:dirtype;var errcode:errtype;medianame:mediatype);external;

procedure reporterr(var idevice,odevice:text;errcode:errtype);

const
  err_nc='ERROR: routine not complete';
  err_nj='ERROR: invalid job or job does not exist';
  err_om='ERROR: out of memory';
  err_or='ERROR: out of range';
  err_bo='ERROR: buffer overflow';
  err_no='ERROR: not open';
  err_nf='ERROR: not found';
  err_ex='ERROR: file already exists';
  err_iu='ERROR: file/device already in use';
  err_ef='ERROR: end of file (EOF)';
```

```
  err_df='ERROR: drive full';
  err_bn='ERROR: bad name';
  err_te='ERROR: transmission error on RS232';
  err_ff='ERROR: format failed';
  err_bp='ERROR: bad parameter';
  err_fe='ERROR: file error';
  err_xp='ERROR: expression error';
  err_ov='ERROR: overflow ,ie when division by ZERO';
  err_ni='ERROR: not implemeneted';
  err_ro='ERROR: read only';
  err_bl='ERROR: bad line';

begin
  case errcode of
  -1: writeln(odevice,err_nc);
  -2: writeln(odevice,err_nj);
  -3: writeln(odevice,err_om);
  -4: writeln(odevice,err_or);
  -5: writeln(odevice,err_bo);
  -6: writeln(odevice,err_no);
  -7: writeln(odevice,err_nf);
  -8: writeln(odevice,err_ex);
  -9: writeln(odevice,err_iu);
 -10: writeln(odevice,err_ef);
 -11: writeln(odevice,err_df);
 -12: writeln(odevice,err_bn);
 -13: writeln(odevice,err_te);
 -14: writeln(odevice,err_ff);
 -15: writeln(odevice,err_bp);
 -16: writeln(odevice,err_fe);
 -17: writeln(odevice,err_xp);
 -18: writeln(odevice,err_ov);
 -19: writeln(odevice,err_ni);
 -20: writeln(odevice,err_ro);
 -21: writeln(odevice,err_bl);
   0: {okay}
 end; {case}

writeln(odevice);
writeln(odevice,'Press ENTER to continue ');
readln;

end; { error proc }

BEGIN {main}
   errcode:=0;
   cnt:=0;
   cnt1:=1;
   write(output,'Enter Ql media name (ie flp1_,etc): ');
```

```
  repeat
    read(input,ch);
    if ch IN
       ['r','a','m','R','A','M','f','l','p','F','L','P','d','v','D','V','1'..'4','_']
    then begin
         cnt:=cnt+1;
         medianame[cnt]:=ch;
        end;
  until (cnt>5) or eoln;

  ramaddr:=addr(dirlist);
  writeln(output,'Ram location of dirlist array BEFORE calling dirql ',ramaddr);
  readln;

  dirql(dirlist,errcode,medianame);

  ramaddr:=addr(dirlist);
  writeln(output,'Ram location of dirlist array AFTER  calling dirql ',ramaddr);
                              {BEFORE=AFTER else corruption}
  writeln(output);

  reporterr(input,output,errcode); {errcode should be EOF}

  If (errcode=0) or (errcode=-10) then {only valid for no error or EOF}
    begin
     for cnt:=0 to (direntries-1) do
      begin
       for cnt1:=1 to dirhdrlen do
        begin
         ch:=dirlist[(cnt*dirhdrlen)+cnt1];
         if (ch=chr(LF)) or (ch=chr(CR)) then ch:=chr(SP);
         write(output,ch);
        end; {for}
       writeln(output);
      end; {for_outer}
   end; {if}

  writeln(output);
  writeln(output,' dirql finished, press ENTER to exit ');
  readln;

END.
```

Appendix 1: PASCAL and its SuperBASIC (SBASIC) translation tables

Table 1: Operators/Expressions

Table 2: Mathematical Functions

Table 3: Identifiers/Types

Table 4: Strings

Table 5: File Commands

Table 6: Screen/Graphic Commands

Table 7: Control Structures

Table 8: Miscellaneous

Table 9: Pascal Data Type Names and Ranges

Note: in the tables below, the SuperBASIC keywords/operators/etc are written in lowercase and the Pascal equivalents in UPPERCASE to differentiate them.

Table 1: Operators/Expressions

SuperBASIC	**Pascal**
=	:=
=	=
==	
+ - * /	+ - * /
div	DIV
mod	MOD
> < <> <= >=	> < <> <= >=
( )	( )
+ (plus)	+ (plus)
- (minus)	- (minus)
&& ^^ -- ||	
and	AND
or	OR
not	NOT
&	CONCAT

Table 2: Mathematical Functions (trigonometry functions in radians)

SuperBASIC	**Pascal**
abs	ABS
acos	-Tan(X+PI/2)
acot	
asin	
atan	ARCTAN
cos	COS
deg	X*PI/180
exp	EXP
int	TRUNC
ln	LN
log	LN/2.303
pi	PI (where PI=3.1415926)
rad	Radians by default
	ROUND
sin	SIN
x^2	SQR
sqrt	SQRT
tan	SIN(X)/COS(X)

Table 3: Identifiers/Types

SuperBASIC	**Pascal**
	D d (IEEE double precision)
E e	E e (scientific notation)
	H (hexadecimal notation)
%	INTEGER MAXINT
$	CHAR TEXT
	BOOLEAN (TRUE FALSE)
Real (by default)	REAL LONGREAL
DIM (matrix) DIMN (matrix)	PACK/UNPACKED/PACKED ARRAY[matrix] of type STRING[strlen]
	^ (pointer type) NEW NIL DISPOSE WITH
	FILE OF type
	SET of type + - * > < >= <= <> IN []
	PACKED RECORD END WITH
LET	Not applicable in Pascal
REMark	{}
'' ""	'' ""

Notes:
In Pascal, type can be INTEGER, CHAR, RECORD, ^, REAL, LONGREAL, BOOLEAN.
In SuperBASIC, the name defines the type, e.g. where % is an integer and $ is a string
matrix - 1,2,3 or more dimensions (e.g. A[2,10] is a two dimensional matrix).
strlen = length of the string

Table 4: Strings

SuperBASIC	**Pascal**
&	CONCAT
bin$ bin	
cdec$ fdat$ fexp$ idec$	WRITE WRITELN
chr$	CHR
code	ORD
	COPY
See QL concepts reference guide, page 16	DELETE
extra$	
fill$	
hex$ hex	
instr	POS
See QL concepts reference guide, page 16	INSERT
len	LENGTH
	PROMPT
STR$ is not required in SuperBASIC, see QL concepts reference guide, page 7	STR

Table 5: File Commands

SuperBASIC	Pascal	SuperBASIC	Pascal
channel_id	HANDLE	INPUT	READ READLN
close	CLOSE	lbytes sbytes	FILE OF 0..255 with RESET with REWRITE
	CHECKFN	merge	{$I}
copy copy_n wcopy wcopy_H wcopy_O		open fopen	ASSIGN UPDATE APPEND
delete wdel	ERASE	open_in for_in	RESET
dir wdir open_dir dup ddown dnext	See listing 9 and 10	open_new for_new	REWRITE
eof		open_out for_out	REWRITE UPDATE
exec exec_w Sexec et ew ex	EXECPROG EXITPROG GETCOMM	open_over for_over	REWRITE UPDATE
datad$ data_use	ASSIGN	parnam$ parstr$ partyp paruse	
dest$ dest_use	ASSIGN	print	WRITE WRITELN
flen	See listing 9/10	progd$ prog_use	ASSIGN
flp_use		put bput	PUT
flush		ram?_	RAMFILE
fnames		rename	RENAME
format	See listing 7 / 8		SEEK
fpos	POSITION	stat wstat	
ftest	FSTAT		TEXTNOTE TEXTPOINT
ftype	OWNERERR	view	ECHO
get_bget	GET		

Table 6: Screen/Graphic Commands

SuperBASIC	Pascal	SuperBASIC	Pascal	SuperBASIC	Pascal
arc arc_r	ARC	flash	FLASH	point	POINT
at	ATC	ink	INK	print print_using	WRITE WRITELN
block	BLOCK	input	READ READLN	recol	RECOL
border	BORDER	line line_r	LINE	scale	SCALE
circle circle_r	CIRCLE	mode	MODE	scroll	SCROLL
cls	CLS	move		strip	STRIP
csize	CSIZE	open	WOPEN	turn	
cursor	ATG ATP	over	OVER	turnto	
cursen curdis	See listing 5 / 6	pan	PAN	under	UNDER
ellipse ellipse_r	ELLIPSE	paper	PAPER	window	WINDOW
fill	FILL	pendown penup			WSTATC WSTATP

Note:

Formatting integer, real and exponential numbers in Pascal, e.g. Formatting the screen output of the keyword PI from 3.1415926 to 3.14 can be achieved using
write(output,'pi=',pi:4:2);
Where 4 signifies the number of characters to output including the decimal point (3.14) and any negative sign (when applicable). The 2 signifies 2 characters after the decimal point, see "Mastering Pascal Programming", page 31-32 by E.Huggins.

Table 7: Control Structures where {} means optional

SuperBASIC	**Pascal**
for x=x1 to x2 step -1: next x	FOR x=x1 DOWNTO X2 do {begin} {end}
for x=x1 to x2 step +1: next x	FOR x=x1 TO X2 do {begin} {end}
for x=x1 to x2 step y: next x for x=x1 to x2 step y: exit : end for x	REPEAT x=x+stepy UNTIL x>x2 WHILE x<=x2 DO begin x=x+stepy; end;
if then else end if	IF THEN {begin} {end} ELSE {begin} {end}

REPeat : exit : END REPeat	exit:=false;WHILE exit=false DO BEGIN SUCC(exit); END;
	exit:=false;REPEAT SUCC(exit);
	UNTIL exit=true;

goto 1000 on gosub 1000	GOTO label (Pascal doesn't use line numbers)
DEFine PROCedure END DEFine	PROCEDURE BEGIN END
DEFine FUNCTion END DEFine	FUNCTION BEGIN END
SELect END SELect	CASE BEGIN OTHERWISE END
Inkey$ keyrow input	COSILENT CONSINGLE CSTAT READ READLN

pause	User defined, see section 1 &/or example below:-
	PROCEDURE
	PAUSE(seconds:-1..mazint);
	VAR loop, temp:-1..maxint;
	Begin
	IF seconds=-1 THEN readln
	ELSE
	FOR loop=0 to seconds DO
	temp:=temp+1;
	{endif}
	END; {pause}

Table 8: Miscellaneous

SuperBASIC	Pascal	SuperBASIC	Pascal	SuperBASIC	Pascal
adate sdate		free_mem	MEMAVAIL	x=x-1 x=x+1	PRED SUCC
	ADDR		HANDLE	randomise	SEED
baud		jobs ajob nxjob ojob pjob spjob		read restore data	See listings 3 & 5
beep beeping			MOVE	respr rechp alchp del_defb extras	
call	QTRAP	net nfs		rnd	RAND
char_inc char_use		See listing 3	ODD	STOP	END.
clock	TIME		PAGE		SIZEOF
date	DATE	peek peek_w peek_l	PEEK See listing 1	spl spl_use	
date$ day$		poke poke_w poke_l	POKE See listing 1	width	

Table 9: Pascal Data Type Names and Ranges with RAM requirements.
NB: ASCII set is 0 to 127 and extended ASCII+ is 0 to 255.

Data Type Name	**Date Range**		**RAM (bytes)**	
	minimum	**maximum**	**minimum**	**maximum**
INTEGER	-214,748,367	2,147,483,647	1	4
REAL (7-bit precision)	E-38	E+38	4	4
LONGREAL (16-bit precision)	E-308	E+308	8	8
BOOLEAN	false	true		1
CHAR	ASCII	ASCII+		1
STRING[n]	1 ASCII char	32767 ASCII+ chars	1	2
SET	1 element	262128 elements	1	2
FILE	text	non-text		
ARRAY	1 dimension	Unlimited dimensions		
enumerated	1 constant	256 constants		1
^ (pointers)				4

Appendix 2: Programs/utilities included with PROPASCAL for the Sinclair QL

Program	Description
PAS	compiler control program
PROPAS1	pass 1 compiler, generates pseudo code
PROPAS2	pass 2 compiler, generates RELocatable code
PROPAS_ERR	error messages
LINK	GST linker
PASLIB_REL	pascal library
PLINT_REL	library start module
PLEND_REL	library end module
PAS_LINK	linker command file
PRL	software Pascal toolkit
PROLIB	librarian
XREF	cross reference generator
PCHECK	ProPascal copy verifier
SETDDEV	change media defaults
SETSTACK	change stack requirements
NOQNS	remove Pascal ? window
TRAPREG_PAS	include file - QTRAP proc
GRAPHIC_PAS	include file - Graphics
COMPARE_PAS	demonstration listing
SQUARE_PAS	demonstration listing
LIFE_PAS	demonstration listing
RESULTS_PAS	demonstration listing

Appendix 3: Boot files for QPAC2 and for the Amiga QL emulator when running ProPascal for the QL.

Sample boot file for QPAC2 users of ProPascal

```
100 REMark - extensions Loaded
110 base=RESPR(14270): LBYTES
    'flp1_ptr_gen',base: CALL base
120 base=RESPR(9992): LBYTES
    'flp1_wman',base: CALL base
130 base=RESPR(10846): LBYTES
    'flp1_hot_rext',base: CALL base
140 base=RESPR(6074): LBYTES
    'flp1_xtras',base: CALL base
144 REMark if you have ROM PRL, then
    miss out line 145
145 base = RESPR(16314): LBYTES
    'mdv1_PRL',base:CALL base+8
150 base=RESPR(34812): LBYTES
    'flp1_Qpac2',base: CALL base
155 REMark TK2_EXT for Trump Card users
160 ERT HOT_RES ('e','flp1_edt_bin','i')
161 ERT HOT_LOAD ('p','flp1_pas')
162 ERT HOT_LOAD ('l','flp1_link')
164 ERT HOT_LOAD ('d','flp1_dectohex')
166 ERT HOT_LOAD ('h','flp1_hextodec')
168 ERT HOT_LOAD ('n','flp1_noqns')
170 ERT HOT_WAKE ('x','Exec')
180 ERT HOT_PICK ('b','')
190 HOT_GO
```

Proposacal Patch for the Amiga

First of all I did not write this patch, I think it was one of the German QL-ers who wrote the QL emulator on the Amiga. As far as I am aware it is public domain. I have include for reference. To work properly, I think you need a ROM image of the ProPascal ROM, which can be achieved using:

SBYTES 'flp1_prlrom',16384,32768

Now type out the listing below and save it to disc as

Boot_ProPas

Note: requires TK2 or disc interface with TK2.

Note: all the files identified on lines 50,1070,1080 and 1090 and the PRL must be stored on the same disc. Now, load using

LRUN flp1_Boot_ProPas

Whenever you want to Pascal source text, copy it to ram1_ and type

PAS "pascal name of code to compile"

This will execute the ProPascal compiler.

```
5 DRV$='FLP1_'
10 RESTORE: REPeat COPLOOP
20 IF EOF THEN EXIT COPLOOP
30 READ I$: PRINT I$: COPY DRV$ & IS$ to 'RAM1_'& I$
40 END REPeat
50 DATA
'PASLIB_REL','PAS_LINK','PLINITQ_REL','PLEND_REL','PROPAS_ERR'
60 PROG_USE 'flp1_' : DATA_USE 'ram1_'
100 PRINT 'to compile a program, load it into ramdisk and type PAS
"name"'
1000 DEFine PROCedure PAS(I$)
1010 LOCal OPT$(80),dev$(5)
1015 POKE HEX("280F2"),10: REMark low screen priority
1020 dev$=DATAD$
1030 OPT$=" NNNNNNNYNNNNNYN"
1040 OPT$=OPT$ & CHR$(0) & CHR$(5) & dev$ & CHR$(0)
1050 OPT$=OPT$ & CHR$(0) & CHR$(21) & dev$ & "TEM$_12345678_IL" &
CHR$(0)
1060 OPT$=OPT$ & CHR$(0) & CHR$(LEN(I$)+5) & dev$ & I$
1070 EW 'PROPAS1':OPT$
1080 EW 'PROPAS2':OPT$
1090 EX 'LINK';dev$ & I$ & ' -PROG ' & dev$ & I$ & ' -NOLIST -WITH '
& dev$ & 'PAS_LINK '
1095 POKE HEX("280F2"),80: REMark high screen priority
1100 END DEFine PAS
```

Appendix 4: SuperBASIC to PC TurboPascal written by Rainer Kowallik

There is reasonable similarity between ProPascal and TurboPascal, so this SuperBASIC program may speed up the conversion process and extend your code for use with PC compatibles. Hopefully Rainer will not mind my inclusion as a reference.

```
1 REMark $$stak=8000
2 REMark $$heap=16000
10 REMark
------------------------------------------------------------
20 REMark SUPERBASIC -> TURBO PASCAL   Compiler
30 REMark
40 REMark This Compiler is intended to reduce the amount of work
50 REMark but not to REPLACE the human work.
60 REMark There are certain differences in the language definition
70 REMark which may cause difficulties. You have to make some affort
by
80 REMark yourself to get the compiled program to work, but this
90 REMark Compiler may reduce stupid typing work to the minimum.
100 REMark Up to now it will do the following steps for you :
105 REMark 0) deleting linenumbers
110 REMark 1) collecting global variables and declaring them at the
Top of
120 REMark       your programm
130 REMark 2) replacing '=' by ':=' and ':' by ';' and '(nn)' by
'[nn]'
135 REMark      and '$' by '_s' and '%' by '_i'
140 REMark 3) defining LOCal variables in the appropriate form
150 REMark 4) Changing 'DEFine FuNction' and 'DEFine PROCedure' into
160 REMark       PASCAL like statements.
165 REMark       Put Procedure arguments in brackets.
170 REMark 5) Converting 'REPeat' and 'FOR' loops
180 REMark 6) processing 'IF' clauses
190 REMark 7) processing 'SELect' statements
200 REMark 8) replacing 'PRINT' by 'WRITE(...)' or 'WRITELN(...)'
210 REMark 9) replacing 'INPUT' by 'WRITE(...) ; READLN(...)'
220 REMark 10) replacing '#n' channel numbers by 'CHNOn' (only 0 to
9)
230 REMark 11) replacing 'nn$( a TO b )' by 'COPY(nn_s,a,b-a)'
232 REMark               'A$ & B$'       by 'CONCAT(A$,B$)'
233 REMark          and  'A$ INSTR B$'   by 'POS(A$,B$)'
235 REMark 12) converting 'REMarks'
237 REMark 13) taking care about GOTO statements
240 REMark
250 REMark WARNING !!! UP TO NOW YOU HAVE TO DECLARE VAR PARAMETERS
BY
260 REMark      YOUR OWN !!!
1000 REMark
------------------------------------------------------------
1010 :
1020 DIM Ivars$(128,32),Rvars$(128,32),Svars$(128,32)
1030 DIM Funcs$(128,32),Procs$(256,32),GOTOlines(128)
1035 DIM Nof_FOR%(128),Nof_REP%(128)
1040 DIM
Forwards$(128,100),AssociateSTOs$(128,32),AssociateSTOn(128)
1050 DIM infil$(80),last_fn$(80),Version$(9),TmpFil$(32)
1060 Version$='1.02' : TmpFil$='RAM1_BasPasTmp'
```

```
1070 Ivarn=0 : Rvarn=0 : Svarn=0 : Funcn=0 : Procn=0 : GOTOn=0
1080 IF_flg=0 : FOR_flg=0 : REP_flg=0 : REPlbCnt=-1
1090 Last_FOR=0 : Last_IF=0 : Forwardn=0 : Associaten=0 :
Main_flg%=1
1100 :
1110 WINDOW #0,448,100,32,142 : BORDER #0,1,0,2 : PAPER #0,0 : INK
#0,5
1120 WINDOW #1,500,112,6,29 : PAPER #1,0 : INK #1,7
1130 WINDOW #2,448,22,32,6 : BORDER #2,1,7 : PAPER #2,0,2 : INK #2,5
1140 CLS #2 : CLS #1 : CLS #0
1150 CSIZE #2,2,1 : PRINT #2,'BasPas  V ';Version$;'     Rainer
Kowallik';
1152 PRINT 'Since EXIT and NEXT are replaced by GOTO statements,'
1154 PRINT 'Label numbers starting from 9960 are reserved !'
1156 PRINT 'If your Program uses GOTO too, you first should RENUM it
!'
1160 AT #1,3,0 : INPUT 'Input file ?',infil$
1170 INPUT 'output file ?',outfil$
1180 PRINT 'Include Sourcecode as comments? (Y/N)'
1190 CURSEN #1 : A$=INKEY$(#1,-1)
1195 Src_Include=1 : Lin_Include=0
1200 IF (A$='N') OR (A$='n') THEN
1202    Src_Include=0 : Lin_Include=1
1204    PRINT 'Include Linenumbers as comments ? (Y/N)' :
A$=INKEY$(#1,-1)
1206    IF (A$='N') OR (A$='n') THEN Lin_Include=0
1208 END IF
1210 Tstart=DATE
1212 PRINT #0,'Prescan (shuffle Mainprogram to end of file)'
1214 Prescan infil$,TmpFil$
1220 PRINT #0,'Pass 1 :.............................'
1225 Init_Keys
1230 Scan_File TmpFil$,1
1240 CLS #1 : PAPER #1,2
1250 PRINT #0,'Pass 2 :.............................'
1260 OPEN_NEW #6,outfil$
1270 PRINT #6,'(* Compiled to PASCAL from SuperBASIC / True BASIC
*)'
1280 PRINT #6,'(* with BasPas Version ';Version$;' by Rainer
Kowallik *)'
1290 PRINT #6,'(* You may need to include the library BasPas.Lib *)'
1300 PRINT #6,'PROGRAM ';infil$;'(INPUT,OUTPUT) ;'
1310 Declare_Globals
1320 Scan_File TmpFil$,2
1330 CLOSE #6 : Tend=DATE : DELETE TmpFil$
1340 CLOSE #1 : OPEN #1,scr_
1350 PAPER #1,0 : CLS #1 : AT #1,3,0
1360 PRINT 'Compilation complete.'\'Time used :',Tend-Tstart;'
seconds'
1370 STOP
2000 REMark
------------------------------------------------------------
2010 REMark defining keywords as procedure to force enclosing in
brackets
2020 REMark
------------------------------------------------------------
2030 DEFine PROCedure Init_Keys
2035 LOCal n,A$(100)
```

```
2040 RESTORE
2050 REPeat Read_Keys
2060    IF EOF THEN EXIT Read_Keys
2070    READ A$
2080    Procs$(Procn)=A$ : Procn=Procn+1
2090 END REPeat Read_Keys
2100 END DEFine Init_Keys
2110 :
2120 DATA 'AT','ARC','ARC_R','BAUD','BEEP','BLOCK','BORDER'
2130 DATA 'CIRCLE','CIRCLE_R','CLEAR','CLOSE','CLS','CSIZE','CURSOR'
2140 DATA
'RESTORE','FILL','FLASH','INK','LINE','LINE_R','MODE','MOVE'
2150 DATA
'OVER','PAN','PAPER','PAUSE','PENUP','PENDOWN','POINT','POKE_W'
2160 DATA 'POKE_L','POKE','RECOL','SCALE','SCROLL','STRIP','TURN'
2170 DATA 'TURNTO','UNDER','WIDTH','WINDOW'
2180 :
3000 REMark
-------------------------------------------------------
3010 REMark Scanning declared and undeclared Variables (FNcode=1)
3020 REMark or processing statements                   (FNcode=2)
3030 REMark
-------------------------------------------------------
3040 DEFine PROCedure Scan_File(FilNam$,FNcode)
3050 LOCal A$(256),B$(256),n,m
3060 Indent$='' : Max_PROC%=0
3070 OPEN_IN #5,FilNam$
3080 REPeat Scan_file_1
3090    IF EOF(#5) THEN EXIT Scan_file_1
3100    INPUT #5,A$ : n=' ' INSTR A$ : ActLine=A$(1 TO n)
3110    A$=A$(n+1 TO LEN(A$)) : ActLine$=A$
3120    PRINT A$
3130    IF FNcode=2 THEN
3140            IF Lin_Include=1 THEN PRINT #6,\'(* ';ActLine;' *)';
3150            IF Src_Include=1 THEN PRINT #6,\'(* ';ActLine$;'
*)';
3155            ProcessNewLine
3160    END IF
3170    REPeat Next_statement
3180            IF A$='' THEN EXIT Next_statement
3190            n=NextDdot(A$)
3200            IF n=LEN(A$) THEN EXIT Next_statement
3210            IF n>0 THEN
3220                    B$=A$(1 TO n-1) : A$=A$(n+1 TO LEN(A$))
3230            ELSE
3240                    B$=A$ : A$=''
3250            END IF
3260            B$=HackBlank$(B$)
3270            IF B$(1 TO 7)='REMark ' THEN
3280                    IF FNcode=2 THEN ProcessRemark B$
3290                    EXIT Next_statement
3300            END IF
3310            IF B$='' THEN NEXT Next_statement
3320            SELect ON FNcode
3330 =1:                Analyse_statement_1 B$
3340 =2:                IF B$(1 TO 4)='DIM ' THEN NEXT
Next_statement
3350                    Analyse_statement_2 B$
```

```
3360            END SELect
3370    END REPeat Next_statement
3380    IF Last_IF=ActLine THEN ProcessEND_IF
3390    IF Last_FOR=ActLine THEN ProcessEND_FOR
3400 END REPeat Scan_file_1
3410 CLOSE #5
3420 IF FNcode=2 THEN PRINT #6,\'9999:'\'END.'
3430 END DEFine Scan_File
3440 :
3450 REMark ------------------------------------------------
3460 REMark     check for hidden variable declaration
3470 REMark     and Procedure / Function definitions
3480 REMark     and GOTO statements
3490 REMark ------------------------------------------------
3500 DEFine PROCedure Analyse_statement_1(A$)
3510 LOCal n,m,B$(256),C$(80)
3520 B$=HackBlank$(A$)
3530 :
3540 IF B$(1 TO 6) = 'GO TO ' THEN
3550    n=B$(7 TO LEN(B$)) : GOTOlines(GOTOn)=n : GOTOn=GOTOn+1 :
RETurn
3560 END IF
3570 :
3580 IF B$(1 TO 16) = 'DEFine FuNction ' THEN
3590    C$=B$(16 TO LEN(B$)) : n='(' INSTR C$
3600    IF n>0 THEN C$=C$(1 TO n-1)
3610    C$=HackBlank$(C$) : Funcs$(Funcn)=C$ : Funcn=Funcn+1
3620    Forwards$(Forwardn)=ProcessFuNction$(B$)
3630    Forwardn=Forwardn+1
3635    Max_PROC%=Max_PROC%+1 : Main_flg%=0
3640 END IF
3650 :
3660 IF B$(1 TO 17) = 'DEFine PROCedure ' THEN
3670    C$=B$(18 TO LEN(B$)) : n='(' INSTR C$
3680    IF n>0 THEN C$=C$(1 TO n-1)
3690    C$=HackBlank$(C$) : Procs$(Procn)=C$ : Procn=Procn+1
3700    Forwards$(Forwardn)=ProcessPROC$(B$)
3710    Forwardn=Forwardn+1
3715    Max_PROC%=Max_PROC%+1 : Main_flg%=0
3720 END IF
3722 :
3724 IF B$(1 TO 11)= 'END DEFine ' THEN Main_flg%=1
3730 :
3740 IF B$(1 TO 4) = 'DIM ' THEN
3750    B$=B$(5 TO LEN(B$)) : B$=HackBlank$(B$)
3760    REPeat scan_dim
3770            n=')' INSTR B$
3780            IF n>0 THEN
3790                    C$=B$(1 TO n)
3800                    IF LEN(B$)>n THEN
3810                            B$=B$(n+1 TO LEN(B$))
3820                    ELSE
3830                            B$=''
3840                    END IF
3850                    n=',' INSTR B$ : IF n>0 THEN B$=B$(n+1 TO
LEN(B$))
3860                    B$=HackBlank$(B$)
3870            ELSE
```

```
3880                    C$=B$
3890            END IF
3900            m=GetType(C$)
3910            InsertVar C$,m
3920            IF n<=0 THEN EXIT scan_dim
3930    END REPeat scan_dim
3940 END IF
3942 :
3945 IF B$(1 TO 3)='IF ' THEN RETurn
3947 :
3950 IF B$(1 TO 7)='REPeat ' THEN
3951    IF Main_flg%=0 THEN
3952       Nof_REP%(Max_PROC%)=Nof_REP%(Max_PROC%)+1
3953    ELSE
3954       Nof_REP%(0)=Nof_REP%(0)+1
3955    END IF
3956    RETurn
3957 END IF
3958 :
3960 IF B$(1 TO 4)='FOR ' THEN
3962    IF Main_flg%=0 THEN
3964       Nof_FOR%(Max_PROC%)=Nof_FOR%(Max_PROC%)+1
3966    ELSE
3968       Nof_FOR%(0)=Nof_FOR%(0)+1
3970    END IF
3972    RETurn
3973 END IF
3974 :
3975 IF B$(1 TO 5)='ELSE ' THEN RETurn
3977 :
3980 n= '=' INSTR B$  : IF n<2 THEN RETurn
3990 C$=B$(1 TO n-1) : C$=HackBlank$(C$)
4000 IF '(' INSTR C$ THEN RETurn
4010 m=GetType(C$) : InsertVar C$,m
4020 END DEFine Analyse_statement_1
4030 :
4040 REMark
------------------------------------------------------------
4050 REMark Function returns string without leading and trailing
blanks
4060 REMark
------------------------------------------------------------
4070 DEFine FuNction HackBlank$(A$)
4080 LOCal n,B$(256)
4090 B$=A$
4100 REPeat HackLead
4110    IF B$(1)<>' ' THEN EXIT HackLead
4120    n=LEN(B$)
4130    IF n<2 THEN EXIT HackLead
4140    B$=B$(2 TO n)
4150 END REPeat HackLead
4160 REPeat HackTrail
4170    n=LEN(B$) : IF n<2 THEN EXIT HackTrail
4180    IF B$(n)<>' ' THEN EXIT HackTrail
4190    B$=B$(1 TO n-1)
4200 END REPeat HackTrail
4210 IF B$=' ' THEN B$=''
4220 RETurn B$
```

```
4230 END DEFine HackBlank$
4240 :
4250 REMark
------------------------------------------------------------
4260 REMark Function returns Type of variable (real=1,%=2,$=3)
4270 REMark
------------------------------------------------------------
4280 DEFine FuNction GetType(A$)
4290 LOCal n,m,B$(256)
4300 B$=A$ : m=1
4310 n='%' INSTR B$
4320 IF n>0 THEN m=2 : RETurn m
4330 n='$' INSTR B$
4340 IF n>0 THEN m=3
4350 RETurn m
4360 END DEFine GetType
4370 :
4380 REMark ------------------------------------------------------------
4390 REMark insert variable in list if not allready present
4400 REMark ------------------------------------------------------------
4410 DEFine PROCedure InsertVar(A$,m)
4420 LOCal n,f
4430 n=m
4440 SELect ON n
4450 =1: f=CheckPresence(A$,Rvars$,Rvarn)
4460     IF f<=0 THEN Rvars$(Rvarn)=A$ : Rvarn=Rvarn+1
4470 =2: f=CheckPresence(A$,Ivars$,Ivarn)
4480     IF f<=0 THEN Ivars$(Ivarn)=A$ : Ivarn=Ivarn+1
4490 =3: f=CheckPresence(A$,Svars$,Svarn)
4500     IF f<=0 THEN Svars$(Svarn)=A$ : Svarn=Svarn+1
4510 END SELect
4520 END DEFine InsertVar
4530 :
4540 REMark
------------------------------------------------------------
4550 REMark check for presence of variable in actual list
4560 REMark
------------------------------------------------------------
4570 DEFine FuNction CheckPresence(A$,VarList$,VarListn)
4580 LOCal n,f,B$(80),C$(80)
4590 f=-1 : B$=NoDimension$(A$)
4600 FOR n=0 TO VarListn
4610    C$=NoDimension$(VarList$(n))
4620    IF B$=C$ THEN f=1 : EXIT n
4630 END FOR n
4640 RETurn f
4650 END DEFine CheckPresence
4660 :
4670 REMark
------------------------------------------------------------
4680 REMark        strip off any dimension from array
4690 REMark
------------------------------------------------------------
4700 DEFine FuNction NoDimension$(A$)
4710 LOCal n,B$(80)
4720 n='(' INSTR A$
4730 IF n<=0 THEN RETurn A$
4740 B$=A$(1 TO n)
```

```
4750 RETurn B$
4760 END DEFine NoDimension$
4770 :
4780 REMark
------------------------------------------------------------
4790 REMark substitute substring in a given string
4800 REMark
------------------------------------------------------------
4810 DEFine PROCedure Substitute(A$,B$,C$)
4820 LOCal n,m,D$(256)
4830 n=A$ INSTR C$ : Sub_flag%=-1
4840 IF n>=1 THEN
4850    IF n>1 THEN D$=C$(1 TO n-1) & B$ : ELSE D$=B$ : END IF
4860    n=n+LEN(A$)
4870    IF LEN(C$)>=n THEN D$=D$ & C$(n TO LEN(C$))
4880    C$=D$ : Sub_flag%=1
4890 END IF
4900 END DEFine Substitute
4910 :
4920 REMark
------------------------------------------------------------
4930 REMark           look for Statements to convert
4940 REMark
------------------------------------------------------------
4950 DEFine PROCedure Analyse_statement_2(A$)
4960 LOCal n,m,B$(256),C$(256)
4970 B$=HackBlank$(A$)
4980 :
4990 IF B$(1 TO 6) = "PRINT " THEN ProcessPrint B$ : RETurn
5000 IF B$(1 TO 6) = "INPUT " THEN ProcessInput B$ : RETurn
5010 IF B$(1 TO 6) = "LOCal " THEN ProcessLocal B$ : RETurn
5020 IF B$(1 TO 16) = "DEFine FuNction " THEN ProcessFunction B$ :
RETurn
5030 IF B$(1 TO 16) = "DEFine PROCedure" THEN ProcessProcedure B$
:RETurn
5040 IF B$(1 TO 8) = "END FOR " THEN ProcessEND_FOR B$ : RETurn
5050 IF B$(1 TO 11) = "END REPeat " THEN ProcessEND_REP B$ : RETurn
5060 IF B$(1 TO 10) = "END DEFine" THEN ProcessEND_DEF B$ : RETurn
5070 IF B$(1 TO 6) = "END IF" THEN ProcessEND_IF B$ : RETurn
5080 IF B$(1 TO 10) = "END SELect" THEN ProcessEND_SEL B$ : RETurn
5090 IF B$(1 TO 8) = "OPEN_NEW" THEN ProcessOPEN B$,'REWRITE' :
RETurn
5100 IF B$(1 TO 7) = "OPEN_IN" THEN ProcessOPEN B$,'RESET' : RETurn
5110 IF B$(1 TO 5) = "OPEN " THEN ProcessOPEN B$,'RESET' : RETurn
5120 IF B$(1 TO 4) = "ELSE" THEN ProcessELSE B$ : RETurn
5130 IF B$(1 TO 4) = "FOR " THEN ProcessFOR B$ : RETurn
5140 IF B$(1 TO 7) = "REPeat " THEN ProcessREP B$ : RETurn
5150 IF B$(1 TO 10) = "SELect ON " THEN ProcessSEL B$ : RETurn
5160 IF B$(1 TO 3) = "IF " THEN ProcessIf B$ : RETurn
5170 IF B$(1 TO 3) = "ON " THEN ProcessON B$ : RETurn
5180 IF B$(1) = "=" THEN ProcessON B$ : RETurn
5190 IF "RETurn" INSTR B$ THEN ProcessRET B$ : RETurn
5200 IF "EXIT " INSTR B$ THEN processEXIT B$ : RETurn
5210 IF "NEXT " INSTR B$ THEN processNEXT B$ : RETurn
5220 :
5230 B$=ConvertTypes$(B$)
5240 n=' ' INSTR B$ : m=0
5250 IF n>0 THEN C$=B$(1 TO n-1) : m=TypeOf(C$)
```

```
5260 IF m=4 THEN
5270    REMark enclose Procedure in brackets
5280    B$=C$ & '(' & B$(n TO LEN(B$)) & ')'
5290 END IF
5300 Substitute '=',':=',B$ : Substitute '#','CHNO',B$
5310 PRINT #6,B$;' ; ';
5320 END DEFine Analyse_statement_2
5330 :
5340 REMark ------------------------------------------------------
5350 REMark return type of Variable (1 = real , 2 = integer , 3 =
string)
5360 REMark or Procedure (4) or function (5) or unknown (0)
5370 REMark ------------------------------------------------------
5380 DEFine FuNction TypeOf(A$)
5390 LOCal n
5400 FOR n= 0 TO Rvarn : IF Rvars$(n)=A$ THEN RETurn 1 : END IF
5410 FOR n= 0 TO Ivarn : IF Ivars$(n)=A$ THEN RETurn 2 : END IF
5420 FOR n= 0 TO Svarn : IF Svars$(n)=A$ THEN RETurn 3 : END IF
5430 FOR n= 0 TO Procn : IF Procs$(n)=A$ THEN RETurn 4 : END IF
5440 FOR n= 0 TO Funcn : IF Funcs$(n)=A$ THEN RETurn 5 : END IF
5450 RETurn 0
5460 END DEFine TypeOf
5470 :
5480 REMark ------------------------------------------------------
5490 REMark  increment indent
5500 REMark ------------------------------------------------------
5510 DEFine PROCedure Inc_Indent
5520 Indent$=Indent$ & '   '
5530 END DEFine Indent$
5540 :
5550 REMark ------------------------------------------------------
5560 REMark  decrement indent
5570 REMark ------------------------------------------------------
5580 DEFine PROCedure Dec_Indent
5590 LOCal n
5600 n=LEN(Indent$)
5610 n=n-3
5620 IF n>0 THEN
5630    Indent$=Indent$(1 TO n)
5640 ELSE
5650    IF n<0 THEN PRINT #0,'Compiler Error: Indentation wrong !'
5660    Indent$=''
5670 END IF
5680 END DEFine Dec_Indent
5690 :
5700 REMark
------------------------------------------------------------
5710 REMark declaring global variables at the top of your programm
5720 REMark
------------------------------------------------------------
5730 DEFine PROCedure Declare_Globals
5740 LOCal n,m,A$(80)
5750 PRINT #6,'(* ----------- Declaring Global Variables.
------------*)'
5760 PRINT #6,'(* You may remove Variables, which are used in
Procedures'
5770 PRINT #6,'   or Functions only, and are declared there *)'
5775 Declare_LABEL 0
```

```
5780 PRINT #6,'VAR'; : Inc_Indent : ProcessNewLine
5790 FOR n=0 TO Ivarn-1
5800    A$=Ivars$(n) : ConvertArrayDeclaration A$
5810    Substitute '%','_i',A$
5820    PRINT #6,A$;TO 30;' INTEGER;'; : ProcessNewLine
5830 END FOR n
5840 FOR n=0 TO Rvarn-1
5850    A$=Rvars$(n) : ConvertArrayDeclaration A$
5860    PRINT #6,A$;TO 30;' REAL;'; : ProcessNewLine
5870 END FOR n
5880 FOR n=0 TO Svarn-1
5890    A$=Svars$(n) : ConvertArrayDeclaration A$
5900    CvtStrArr A$
5910    PRINT #6,A$; : ProcessNewLine
5920 END FOR n
5930 PRINT #6,'(* declaring ALL FUNCTIONS and PROCEDURES as FORWARD
*)'
5940 ProcessNewLine
5950 FOR n=0 TO Forwardn-1
5960    A$=Forwards$(n)
5970    PRINT #6,A$;' FORWARD;'; : ProcessNewLine
5980 END FOR n
5990 Dec_Indent
6000 END DEFine Declare_Globals
6010 :
6020 REMark -----------------------------------------------------
6030 REMark Find next relevant ':' in actual programm
6040 REMark -----------------------------------------------------
6050 DEFine FuNction NextDdot(A$)
6060 LOCal n,m,flag,B$(9)
6070 m=LEN(A$) : n=0 : flag=0
6080 REPeat findNextDd
6090    n=n+1 : IF n>m THEN n=0 : EXIT findNextDd
6100    B$=A$(n) : IF B$=':' THEN EXIT findNextDd
6110    IF (B$='"') OR (B$="'") THEN
6120            REPeat FindEndStr
6130                    n=n+1 : IF n>m THEN : EXIT FindEndStr
6140                    IF A$(n)=B$ THEN EXIT FindEndStr
6150            END REPeat FindEndStr
6160    END IF
6170 END REPeat findNextDd
6180 RETurn n
6190 END DEFine NextDdot
6200 :
6210 DEFine PROCedure Associate(A$,n)
6220 LOCal i,m,flag
6230 flag=1
6240 FOR i=0 TO Associaten
6250    IF AssociateSTOs$(i)=A$ THEN
6260            flag=0 : AssociateSTOn(i)=n
6270            EXIT i
6280    END IF
6290 END FOR i
6300 IF flag=1 THEN
6310    AssociateSTOs$(Associaten)=A$
6320    AssociateSTOn(Associaten)=n
6330    Associaten=Associaten+1
6340 END IF
```

```
6350 END DEFine Associate
6360 :
6370 DEFine FuNction AssociateNR(A$)
6380 LOCal m,i,flag
6390 flag=1
6400 FOR i=0 TO Associaten
6410    IF AssociateSTOs$(i)=A$ THEN
6420            flag=0 : m=AssociateSTOn(i)
6430            EXIT i
6440    END IF
6450 END FOR i
6460 IF flag=1 THEN m=-1E99
6470 RETurn m
6480 END DEFine AssociateNR
6490 :
6500 REMark
------------------------------------------------------------
6510 REMark Convert A$(A TO B) to COPY(A$,A,B-A)
6520 REMark (may be changed to midstr)
6530 REMark
------------------------------------------------------------
6540 DEFine FuNction CVT_MID$(B$)
6550 LOCal A$(100),C$(100),D$(100),n,m
6560 n='(' INSTR B$ : IF n<=1 THEN RETurn B$
6570 A$=B$(1 TO n-1) : A$=HackBlank$(A$)
6580 m=' TO ' INSTR B$ : IF m<1 THEN RETurn B$
6590 C$=B$(n+1 TO m-1) : C$=HackBlank$(C$)
6600 IF C$='' THEN C$='1'
6610 n=find_ket%(B$,0) : D$='LENGTH(' & A$ & ')'
6620 IF n>0 THEN D$=B$(m+4 TO n-1)
6630 IF C$<>'1' THEN
6640    A$='COPY(' & A$ & ',' & C$ & ',' & D$ & '-' & C$ & '+1)'
6650    ELSE
6660    A$='COPY(' & A$ & ',1,' & D$ & ')'
6670    END IF
6680 RETurn A$
6690 END DEFine CVT_MID$
6700 :
6710 REMark
------------------------------------------------------------
6720 REMark Convert $,%,(),MID$,INSTR in Expression
6730 REMark
------------------------------------------------------------
6740 DEFine FuNction ConvertTypes$(D$)
6750 LOCal A$(100),B$(100),C$(100),E$(100),n,m,i,l,k
6760 B$=D$ : E$=B$ : A$=''
6770 n='=' INSTR B$ : IF n>0 THEN E$=B$(n+1 TO LEN(B$)) : A$=B$(1 TO
n)
6780 IF LEN(D$)>99 THEN PRINT #0,ActLine$ : RETurn ""
6790 IF ' INSTR ' INSTR B$ THEN B$=A$ & CVT_INSTR$(E$)
6800 n=1 : m=0 : l=LEN(B$)
6810 REMark isolate VAR or FN (without indices)
6820 REPeat Nxt_Var
6830    m=m+1 : IF m>=l THEN k=0 : EXIT Nxt_Var
6840    IF (m+1)<l THEN
6850        C$=B$(m TO m+1)
6860        IF (C$='<>') OR (C$='>=') OR (C$='<=') THEN k=7 : EXIT
Nxt_Var
```

```
6870        IF (C$='||') OR (C$='&&') OR (C$='^^') THEN k=7 : EXIT
Nxt_Var
6880        IF (C$='$(') AND (' TO ' INSTR B$) THEN
6890            B$=CVT_MID$(B$) : l=LEN(B$) : m=1
6900        END IF
6910    END IF
6920    C$=B$(m) : IF (C$=',') OR (C$=';') THEN k=1 : EXIT Nxt_Var
6930    IF C$=' ' THEN
6940       IF (m+4)<l THEN C$=B$(m TO m+4)
6950       k=6 : IF C$=' TO ' THEN k=4
6960       IF C$=' OR ' THEN k=7
6970       IF (m+5)<l THEN C$=B$(m TO m+5)
6980       IF (C$=' DIV ') OR (C$=' MOD ') THEN k=7
6990       IF (C$=' AND ') OR (C$=' XOR ') THEN k=7
7000       EXIT Nxt_Var
7010    END IF
7020    IF C$='(' THEN k=2 : EXIT Nxt_Var
7030    IF C$=')' THEN k=3 : EXIT Nxt_Var
7040    IF (C$='+') OR (C$='-') THEN k=7 : EXIT Nxt_Var
7050    IF (C$='*') OR (C$='/') OR (C$='^') THEN k=7 : EXIT Nxt_Var
7060    IF (C$='&') THEN k=7 : Conv_Flg=1 : C$=',' : EXIT Nxt_Var
7070    IF (C$='<') OR (C$='>') THEN k=7 : EXIT Nxt_Var
7080    IF (C$='=') THEN k=7 : Conv_Flg=0 : EXIT Nxt_Var
7090    IF (C$='"') OR (C$="'") THEN
7100            B$(m)="'"
7110            REPeat FindEndStr
7120               m=m+1 : IF m>l THEN k=0 : EXIT Nxt_Var
7130               IF B$(m)=C$ THEN k=5 : B$(m)="'" : EXIT Nxt_Var
7140            END REPeat FindEndStr
7150    END IF
7160 END REPeat Nxt_Var
7170 SELect ON k
7180    =0: A$=Cvt_Var$(B$)
7190    =1: A$='' : E$=''
7200        IF m>1 THEN A$=B$(1 TO m-1) : A$=ConvertTypes$(A$)
7210        IF m<l THEN E$=B$(m+1 TO l) : E$=ConvertTypes$(E$)
7220        A$=A$ & C$ & E$
7230    =2: C$='' : A$='' : E$='' : n=1 : i=m
7240        i=find_ket%(B$,0)
7250        IF i=0 THEN
7260            IF (m<l) THEN C$=B$(m+1 TO l)
7270            ELSE
7280            IF (m<l) THEN C$=B$(m+1 TO i-1)
7290            IF (i<l) THEN E$=B$(i+1 TO l)
7300        END IF
7310        C$=ConvertTypes$(C$)
7320        E$=ConvertTypes$(E$)
7330        IF m>1 THEN A$=B$(1 TO m-1) : A$=Cvt_Var$(A$)
7340        IF Var_Flg=1 THEN
7350            A$=A$ & '[' & C$ & ']' & E$
7360            ELSE
7370            A$=A$ & '(' & C$ & ')' & E$
7380        END IF
7390    =6: C$='' : A$=''
7400        IF m<l THEN C$=B$(m+1 TO l) : C$=ConvertTypes$(C$)
7410        IF m>1 THEN A$=B$(1 TO m-1) : A$=ConvertTypes$(A$)
7420        A$=A$ & ' ' & C$
7430    =3: IF m>1 THEN
```

```
7440           A$=B$(1 TO m-1) : A$=Cvt_Var$(A$)
7450           ELSE
7460           A$=B$(1)
7470        END IF
7480        IF Var_Flg=1 THEN A$=A$ & ']' : ELSE A$=A$ & ')'
7490    =4: A$=B$(1 TO m) : C$=B$(m+4 TO l)
7500        A$=ConvertTypes$(A$) : C$=ConvertTypes$(C$)
7510        A$=A$ & ' TO ' & C$ : A$=CVT_MID$(A$)
7520    =7: REMark +-*/ DIV MOD AND XOR OR
7530        E$=B$(m+LEN(C$) TO l) : A$='' : IF m>1 THEN A$=B$(1 TO
m-1)
7540        A$=ConvertTypes$(A$) : E$=ConvertTypes$(E$)
7550        IF (C$='=') AND (Conv_Flg=1) THEN E$='CONCAT(' & E$ &
')'
7560        A$=A$ & C$ & E$
7570    =5: A$='' : C$=B$(1 TO m)
7580        IF m<l THEN A$=B$(m+1 TO l) : A$=ConvertTypes$(A$)
7590        A$=C$ & A$
7600 END SELect
7610 RETurn A$
7620 END DEFine ConvertTypes$
7630 :
7640 DEFine FuNction Cvt_Var$(A$)
7650 LOCal B$(100),n,m
7660 B$=A$ : Var_Flg=-1
7670 n=TypeOf(B$)
7680 SELect ON n
7690    =1: Var_Flg=1
7700    =2: Var_Flg=1
7710        Substitute '%','_i',B$
7720    =3: Var_Flg=1
7730        Substitute '$','_s',B$
7740    =5: Var_Flg=-1
7750        Substitute '$','_s',B$ : Substitute '%','_i',B$
7760 END SELect
7770 IF B$='LEN' THEN B$='LENGTH'
7780 RETurn B$
7790 END DEFine Cvt_Var$
7800 :
7810 REMark
-------------------------------------------------------------
7820 REMark Find closing bracket in a given string
7830 REMark
-------------------------------------------------------------
7840 DEFine FuNction find_ket%(A$,ii)
7850 LOCal n%,i%,l%
7860 i%=ii : n%=1 : l%=LEN(A$)
7870 REPeat find_ket_lp
7880    IF n%>l% THEN RETurn 0
7890    IF A$(n%)='(' THEN i%=i%+1
7900    IF A$(n%)=')' THEN
7910       i%=i%-1 : IF i%=0 THEN EXIT find_ket_lp
7920    END IF
7930    n%=n%+1
7940 END REPeat find_ket_lp
7950 RETurn n%
7960 END DEFine find_ket%
7970 :
```

```
7980 REMark -------------------------------------------------------
7990 REMark Converting array declaration for Strings
8000 REMark -------------------------------------------------------
8010 DEFine PROCedure CvtStrArr(A$)
8020 LOCal B$(100),C$(100)
8030 B$=A$ : Substitute '$','_s',B$ : Substitute ' OF ','',B$
8040 IF "," INSTR B$ THEN
8050    Substitute ',0..',',] OF STRING[',B$
8060 ELSE
8070    Substitute ' ARRAY ',' STRING',B$
8080    Substitute '[0..','[',B$
8090 END IF
8100 A$=B$
8110 END DEFine CvtStrArr
8120 :
8130 REMark
-----------------------------------------------------
8140 REMark Reading Superbasic program and sort out Main program
8150 REMark Marking Start of Mainprogram with REMark $$MAIN
8160 REMark
-----------------------------------------------------
8170 DEFine PROCedure Prescan(infil$,outfil$)
8180 LOCal A$(256),B$(256),MainProg$(128,128),l,m,n,Count
8190 OPEN_IN #5,infil$ : OPEN_NEW #6,outfil$
8200 n=0 : Count=0
8210 REPeat Copy_procs
8220    IF EOF(#5) THEN  EXIT Copy_procs
8230    INPUT #5,A$ : PRINT A$
8240    l=LEN(A$) : m=' ' INSTR A$ : B$=A$(m+1 TO l) :
B$=HackBlank$(B$)
8250    IF B$(1 TO 16)='DEFine PROCedure' THEN Count=Count+1
8260    IF B$(1 TO 15)='DEFine FuNction' THEN Count=Count+1
8270    IF B$(1 TO 6)='REMark' THEN PRINT #6,A$ : NEXT Copy_procs
8280    IF B$=':' THEN PRINT #6,A$ : NEXT Copy_procs
8290    IF B$(1 TO 10)='END DEFine' THEN
8300       PRINT #6,A$ : Count=Count-1 : NEXT Copy_procs
8310    END IF
8320    IF Count<0 THEN Count=0 : PRINT #0,'Prescan error before'\A$
8330    IF Count=0 THEN
8340       MainProg$(n)=A$ : n=n+1
8350    ELSE
8360       PRINT #6,A$
8370    END IF
8380 END REPeat Copy_procs
8390 :
8400 PRINT #6,'99999 ';'REMark $$MAIN'
8410 FOR m=0 TO n-1
8420    PRINT #6,MainProg$(m)
8430 END FOR m
8440 CLOSE #5 : CLOSE #6
8450 END DEFine Prescan
8460 :
10000 REMark
-----------------------------------------------------------
10010 REMark process new line in BASIC programm
10020 REMark look for GO TO , print indent
10030 REMark
-----------------------------------------------------------
```

```
10040 DEFine PROCedure ProcessNewLine
10050 LOCal n
10060 PRINT #6
10070 FOR n=0 TO GOTOn
10080   IF GOTOlines(n)=ActLine THEN PRINT #6,ActLine;':'
10090 END FOR n
10100 PRINT #6,Indent$;
10110 END DEFine ProcessNewLine
10120 :
10130 REMark
------------------------------------------------------------
10140 REMark processing REMarks
10150 REMark
------------------------------------------------------------
10160 DEFine PROCedure ProcessRemark(A$)
10170 LOCal B$(256),l%
10180 l%=LEN(A$) : B$=''
10190 IF l%>7 THEN B$=A$(7 TO l%)
10200 IF B$=' $$MAIN' THEN PRINT #6,'BEGIN'
10210 B$='(*' & B$ & ' *)'
10220 PRINT #6,B$;
10230 END DEFine ProcessRemark
10240 :
10250 REMark
------------------------------------------------------------
10260 REMark declare local variables
10270 REMark
------------------------------------------------------------
10280 DEFine PROCedure ProcessLocal(A$)
10290 LOCal B$(256),C$(80),n,m
10300 B$=A$(7 TO LEN(A$))
10310 PRINT #6,'VAR'; : Inc_Indent : ProcessNewLine
10320 REPeat ScanLocalVars
10330   C$=NextVar$(B$) : IF C$='' THEN EXIT ScanLocalVars
10340   m=GetType(C$) : ConvertArrayDeclaration C$
10350   SELect ON m
10360 =1: PRINT #6,C$;TO 30;' REAL ;';
10370 =2: Substitute '%','_i',C$ : PRINT #6,C$;TO 30;' INTEGER;';
10380 =3: CvtStrArr C$ : PRINT #6,C$;
10390   END SELect
10400   ProcessNewLine
10410 END REPeat ScanLocalVars
10420 Dec_Indent
10430 END DEFine ProcessLocal
10440 :
10450 REMark
------------------------------------------------------------
10460 REMark get next VARiable name in string and cut string
10470 REMark
------------------------------------------------------------
10480 DEFine FuNction NextVar$(A$)
10490 LOCal n,flag,B$(256)
10500 n=0 : B$='' : flag=0
10510 REPeat NextVarLoop
10520   n=n+1 : IF n> LEN(A$) THEN B$=A$ : A$='' : RETurn B$
10530   IF A$(n)='(' THEN flag=flag+1
10540   IF A$(n)=')' THEN flag=flag-1
10550   IF (A$(n)=',') AND (flag=0) THEN EXIT NextVarLoop
```

```
10560 END REPeat NextVarLoop
10570 B$=A$(1 TO n-1)
10580 IF n<LEN(A$) THEN A$=A$(n+1 TO LEN(A$)) : ELSE A$='' : END IF
10590 RETurn B$
10600 END DEFine NextVar$
10610 REMark
------------------------------------------------------------
10620 REMark convert Array declaration a(n) to a[0..n]
10630 REMark
------------------------------------------------------------
10640 DEFine PROCedure ConvertArrayDeclaration(A$)
10650 LOCal n,flag,aflag,B$(256)
10660 n=0 : B$='' : aflag=0
10670 REPeat convArr
10680   n=n+1 : flag=0 : IF n> LEN(A$) THEN EXIT convArr
10690   IF A$(n)='(' THEN B$=B$ & ' : ARRAY [0..' : flag=1 : aflag=1
10700   IF A$(n)=')' THEN B$=B$ & '] OF '    : flag=1 : EXIT convArr
10710   IF A$(n)=',' THEN B$=B$ & ',0..' : flag=1
10720   IF flag=0 THEN B$=B$ & A$(n)
10730 END REPeat convArr
10740 A$=B$ : IF aflag=0 THEN A$=A$ & ' : '
10750 END DEFine ConvertArrayDeclaration
10760 :
10770 REMark ------------------------------------------------
10780 REMark process PRINT
10790 REMark ------------------------------------------------
10800 DEFine PROCedure ProcessPrint(B$)
10810 LOCal n,m,C$(256)
10820 C$=B$(6 TO LEN(B$))
10830 Substitute '#','CHNO',C$  : C$=ConvertTypes$(C$)
10840 IF B$(LEN(B$))=';' THEN
10850   PRINT #6,'WRITE(';C$;');';
10860 ELSE
10870   PRINT #6,'WRITELN(';C$;');';
10880 END IF
10890 END DEFine ProcessPrint
10900 :
10910 DEFine PROCedure ProcessInput(B$)
10920 LOCal n,m,CHNO$(9),A$(256),C$(256)
10930 C$=B$(6 TO LEN(B$))
10940 C$=ConvertTypes$(C$) : n='#' INSTR C$
10950 PRINT C$ : CHNO$='' : A$=''
10960 IF n>0 THEN
10970   CHNO$='CHNO' & C$(n+1) & ','
10980   n=',' INSTR C$ : C$=C$(n+1 TO LEN(C$))
10990 END IF
11000 n='"' INSTR C$ : IF n=0 THEN n="'" INSTR C$
11010 IF n>0 THEN
11020   C$=C$(n+1 TO LEN(C$))
11030   n='"' INSTR C$ : IF n=0 THEN n="'" INSTR C$
11040   A$=C$(1 TO n-1) : n=',' INSTR C$
11050   C$=C$(n+1 TO LEN(C$))
11060 END IF
11070 IF A$<>'' THEN PRINT #6,"WRITE(";CHNO$;"'";A$;"'); ";
11080 PRINT #6,'READLN(';CHNO$;C$;'); ';
11090 END DEFine ProcessInput
11100 :
11110 DEFine FuNction ProcessFuNction$(B$)
```

```
11120 LOCal A$(256),C$(256),n,m,typ
11130 C$=B$(17 TO LEN(B$)) : n='(' INSTR C$
11140 IF n>0 THEN
11150   A$=C$(1 TO n-1) : C$=C$(n TO LEN(C$))
11160   ELSE
11170   A$=C$ : C$='()'
11180 END IF
11190 typ=1
11200 Substitute '$','_s',A$ : IF Sub_flag%=1 THEN typ=3
11210 Substitute '%','_i',A$ : IF Sub_flag%=1 THEN typ=2
11220 last_fn$=A$
11230 C$=DeclareArgs$(C$)
11240 A$='FUNCTION ' & A$ & C$ & ': '
11250 SELect ON typ
11260   =1: A$ = A$ & 'REAL; '
11270   =2: A$ = A$ & 'INTEGER; '
11280   =3: A$ = A$ & 'STRING; '
11290 END SELect
11300 RETurn A$
11310 END DEFine ProcessFuNction$
11320 :
11330 DEFine FuNction DeclareArgs$(B$)
11340 LOCal A$(256),C$(256),D$(256),n,m,typ
11350 A$='(' : n='(' INSTR B$ : IF n=0 THEN RETurn '()'
11360 C$=B$(n+1 TO LEN(B$))
11370 REPeat scan_args
11380   n=',' INSTR C$ : IF n=0 THEN n=')' INSTR C$
11390   IF n<=1 THEN EXIT scan_args
11400   D$=C$(1 TO n-1)
11410   IF n<LEN(C$) THEN C$=C$(n+1 TO LEN(C$)) : ELSE C$=''
11420   typ=1
11430   Substitute '$','_s',D$ : IF Sub_flag%=1 THEN typ=3
11440   Substitute '%','_i',D$ : IF Sub_flag%=1 THEN typ=2
11450   SELect ON typ
11460      =1: D$=D$ & ':REAL'
11470      =2: D$=D$ & ':INTEGER'
11480      =3: D$=D$ & ':STRING'
11490   END SELect
11500   A$=A$ & D$ & ','
11510 END REPeat scan_args
11520 A$(LEN(A$))=')'
11530 RETurn A$
11540 END DEFine DeclareArgs$
11550 :
11560 DEFine PROCedure ProcessFunction(B$)
11570 LOCal C$(256),n
11575 Max_PROC%=Max_PROC%+1
11580 C$=B$(16 TO LEN(B$)) : n='(' INSTR C$
11590 IF n>0 THEN C$=C$(1 TO n-1)
11600 C$=HackBlank$(C$)
11610 Substitute '$','_s',C$ : Substitute '%','_i',C$
11620 PRINT #6,'FUNCTION ';C$;'; ';
11630 Inc_Indent : ProcessNewLine
11635 Declare_LABEL Max_PROC%
11720 REPlbCnt=-1 : last_fn$=C$
11730 END DEFine ProcessFunction
11740 :
11750 DEFine FuNction ProcessPROC$(B$)
```

```
11760 LOCal A$(256),C$(256),n,m,typ
11770 C$=B$(17 TO LEN(B$)) : n='(' INSTR C$
11780 IF n>0 THEN
11790   A$=C$(1 TO n-1) : C$=C$(n TO LEN(C$))
11800   ELSE
11810   A$=C$ : C$='()'
11820 END IF
11830 C$=DeclareArgs$(C$)
11840 A$='PROCEDURE ' & A$ & C$ & '; '
11850 RETurn A$
11860 END DEFine ProcessPROC$
11870 :
11880 DEFine PROCedure ProcessProcedure(B$)
11890 LOCal C$(256),n
11895 Max_PROC%=Max_PROC%+1
11900 C$=B$(18 TO LEN(B$)) : n='(' INSTR C$
11910 IF n>0 THEN C$=C$(1 TO n-1)
11920 C$=HackBlank$(C$)
11930 PRINT #6,'PROCEDURE ';C$;'; ';
11940 Inc_Indent : ProcessNewLine
11950 Declare_LABEL Max_PROC%
12030 REPlbCnt=-1
12040 END DEFine ProcessProcedure
12041 :
12042 DEFine PROCedure Declare_LABEL(n%)
12044 LOCal i,k
12046 PRINT #6,'LABEL'
12048 k=Nof_FOR%(n%)-1 : PRINT #6,'      ';
12050 FOR i=0 TO k :  PRINT #6,'997';i;',996';i;',';
12052 k=Nof_REP%(n%)-1 : PRINT #6,\'      ';
12054 FOR i=0 TO k :  PRINT #6,'999';i;',998';i;',';
12056 PRINT #6,\'      9999;'
12058 END DEFine Declare_LABEL
12059 :
12060 DEFine PROCedure ProcessRET(B$)
12070 LOCal A$(256),n,m
12080 n=LEN(B$)
12090 IF LEN(B$)<7 THEN
12100   PRINT #6,'GOTO 9999;';
12110   ELSE
12120   A$=B$(7 TO LEN(B$))
12130   PRINT #6,last_fn$;':=';A$;' ; GOTO 9999 ;';
12140 END IF
12150 END DEFine ProcessRET
12160 :
12170 DEFine PROCedure ProcessEND_FOR(B$)
12180 Dec_Indent : ProcessNewLine : FOR_flg=FOR_flg-1 : Last_FOR=0
12190 PRINT #6,'997';FOR_flg;':'; : ProcessNewLine
12200 PRINT #6,'END; (* FOR *) '; : ProcessNewLine
12210 PRINT #6,'996';FOR_flg;':';
12220 IF FOR_flg<0 THEN
12230   PRINT #0,'Compiler error in FOR loop processing',ActLine
12240   FOR_flg=0
12250 END IF
12260 END DEFine ProcessEND_FOR
12270 :
12280 DEFine PROCedure ProcessEND_REP(B$)
12290 LOCal n,A$(256)
```

```
12300 A$=B$(12 TO LEN(B$)) : A$=HackBlank$(A$)
12310 n=AssociateNR(A$)
12320 Dec_Indent
12330 PRINT #6,'999';n;':'; : ProcessNewLine
12340 PRINT #6,'UNTIL FALSE; (* ';B$;' *)'; : ProcessNewLine
12350 PRINT #6,'998';n;':';
12360 END DEFine ProcessEND_REP
12370 :
12380 DEFine PROCedure ProcessEND_DEF(B$)
12390 PRINT #6,'9999:'; : ProcessNewLine
12400 Dec_Indent
12410 PRINT #6,'END; (* ';B$;' *)';
12420 END DEFine ProcessEND_DEF
12430 :
12440 DEFine PROCedure ProcessEND_IF(B$)
12450 Dec_Indent : ProcessNewLine
12460 PRINT #6,'END; (* IF *)';
12470 ProcessNewLine
12480 IF_flg=IF_flg-1 : Last_IF=0
12490 IF IF_flg<0 THEN
12500   PRINT #0,'Compiler error in processing of IF clause',ActLine
12510   IF_flg=0
12520 END IF
12530 END DEFine ProcessEND_IF
12540 :
12550 DEFine PROCedure ProcessEND_SEL(B$)
12560 PRINT #6,'END;'; : Dec_Indent : ProcessNewLine
12570 PRINT #6,'END; (* SELect *)';
12580 END DEFine ProcessEND_SEL
12590 :
12600 DEFine PROCedure ProcessELSE(B$)
12610 ProcessNewLine
12620 PRINT #6,'ELSE';
12630 ProcessNewLine
12640 END DEFine ProcessELSE
12650 :
12660 DEFine PROCedure ProcessFOR(B$)
12670 LOCal A$(100),Indx$(32),Strt$(32),Limit$(32),Incr$(32),n,m,l
12680 A$=B$ : l=LEN(A$)
12690 n='=' INSTR A$ : Indx$=A$(5 TO n-1) : Indx$=HackBlank$(Indx$)
12700 m=' TO ' INSTR A$ : Strt$=A$(n+1 TO m)
12710 Incr$='1.0' : n=' STEP' INSTR A$
12720 IF n>0 THEN Incr$=A$(n+5 TO l) : ELSE n=l : END IF
12730 Limit$=A$(m+4 TO n)
12740 n=A$ INSTR ActLine$ : m=LEN(ActLine$)
12750 IF n+l<m THEN Last_FOR=ActLine
12760 PRINT #6,Indx$;':=';Strt$;'-';Incr$;' ;'; : ProcessNewLine
12770 PRINT #6,'WHILE ';Indx$;'<';Limit$;' DO'; : ProcessNewLine
12780 Inc_Indent : PRINT #6,'BEGIN'; : ProcessNewLine
12790 PRINT #6,Indx$;':=';Indx$;'+';Incr$;' ;'; : ProcessNewLine
12800 Associate Indx$,100+FOR_flg
12810 FOR_flg=FOR_flg+1
12820 END DEFine ProcessFOR
12830 :
12840 DEFine PROCedure ProcessREP(B$)
12850 LOCal A$(256),n
12860 A$=B$(8 TO LEN(B$)) : A$=HackBlank$(A$)
12870 PRINT #6,'REPEAT (* ';A$;' *)';
```

```
12880 Inc_Indent
12890 ProcessNewLine
12900 REPlbCnt=REPlbCnt+1 : Associate A$,REPlbCnt
12910 END DEFine ProcessREP
12920 :
12930 DEFine PROCedure ProcessSEL(B$)
12940 LOCal A$(100)
12950 A$=B$(11 TO LEN(B$)) : A$=HackBlank$(A$)
12960 PRINT #6,'CASE ';A$;' OF';
12970 ProcessNewLine
12980 Inc_Indent : SEL_flg=0
12990 END DEFine ProcessSEL
13000 :
13010 DEFine PROCedure ProcessIf(B$)
13020 LOCal A$(100),C$(100),n,m,k
13030 ProcessNewLine
13040 m=LEN(B$) : n=' THEN' INSTR B$
13050 A$=B$(4 TO n) : A$=ConvertTypes$(A$)
13060 PRINT #6,'IF ';A$,'THEN BEGIN '; : Inc_Indent : ProcessNewLine
13070 IF_flg=IF_flg+1
13080 IF m>n+5 THEN
13090   A$=B$(n+6 TO m)
13100   Analyse_statement_2 A$ : Last_IF=ActLine
13110 END IF
13120 END DEFine ProcessIf
13130 :
13140 DEFine PROCedure ProcessON(B$)
13150 LOCal A$(100),n,m
13160 n='=' INSTR B$
13170 A$=B$(n+1 TO LEN(B$)) : A$=HackBlank$(A$)
13180 IF A$='REMAINDER' THEN A$='ELSE' : REMark may be changed to
OTHERS
13190 IF SEL_flg=1 THEN PRINT #6,'END;'; : ProcessNewLine
13200 SEL_flg=1
13210 PRINT #6,A$;' : BEGIN'; : ProcessNewLine
13220 END DEFine ProcessON
13230 :
13240 DEFine FuNction CVT_INSTR$(B$)
13250 LOCal A$(100),n
13260 n=' INSTR ' INSTR B$
13270 A$=B$(1 TO n) &',' & B$(n+7 TO LEN(B$))
13280 Substitute '$','_s',A$ : Substitute '$','_s',A$
13290 C$='POS(' & A$ & '); ' : REMark may be changed to "instr"
13300 RETurn C$
13310 END DEFine CVT_INSTR$
13320 :
13330 DEFine PROCedure processEXIT(B$)
13340 LOCal A$(100),C$(100),n,m
13350 n="EXIT " INSTR B$
13360 A$=B$(n+5 TO LEN(B$)) : A$=HackBlank$(A$)
13370 n=AssociateNR(A$)
13380 IF n<0 THEN
13390   PRINT #0,"Compiler error in EXIT processing",A$,ActLine
13400   PRINT #6,'(* ';B$;' *)';
13410   ELSE
13420   C$='998'
13430   IF n>=100 THEN n=n-100 : C$='996'
13440   PRINT #6,'GOTO ';C$;n;' ;';
```

```
13450 END IF
13460 END DEFine processEXIT
13470 :
13480 DEFine PROCedure processNEXT(B$)
13490 LOCal A$(100),C$(100),n,m
13500 n="NEXT " INSTR B$
13510 A$=B$(n+5 TO LEN(B$)) : A$=HackBlank$(A$)
13520 n=AssociateNR(A$)
13530 IF n<0 THEN
13540   PRINT #0,"Compiler error in NEXT processing",A$,ActLine
13550   PRINT #6,'(* ';B$;' *)';
13560   ELSE
13570   C$='999'
13580   IF n>=100 THEN n=n-100 : C$='997'
13590   PRINT #6,'GOTO ';C$;n;' ;';
13600 END IF
13610 END DEFine processNEXT
13620 :
13630 DEFine PROCedure ProcessOPEN(B$,C$)
13640 LOCal A$(100),D$(100),n,m,l
13650 A$=B$ : l=LEN(A$)
13660 n=' ' INSTR A$ : D$=A$(n+1 TO l)
13670 D$=ConvertTypes$(D$) : Substitute '#','CHNO',D$
13680 n=',' INSTR D$ : A$=D$(1 TO n-1)
13690 PRINT #6,'ASSIGN(';D$;') ; ';C$;'(';A$;') ;';
13700 END DEFine ProcessOPEN
13710 :
20000 DEFine PROCedure sve
20010 A$='FLP2_BasPas_BAS'
20020 DELETE A$ : SAVE A$
20030 END DEFine sve
```