The material presented in this document has been expressly prepared by POLYCORP New Zealand Limited.

No part of the publication may be reproduced, stored in a retrieval system, transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior permission of POLYCORP New Zealand Limited.

COPYRIGHT AUGUST 1984

POLYCORP NEW ZEALAND LIMITED
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. PREPARING A PASCAL PROGRAM</td>
<td>2</td>
</tr>
<tr>
<td>3. STANDARD PASCAL</td>
<td>3</td>
</tr>
<tr>
<td>4. TRYING OUT PASCAL</td>
<td>5</td>
</tr>
<tr>
<td>5. OMEGASOFT PASCAL ON POLY</td>
<td>7</td>
</tr>
<tr>
<td>5.1. Extensions to Omegasoft Pascal</td>
<td>7</td>
</tr>
<tr>
<td>5.2. Conversion of Existing Programs</td>
<td>8</td>
</tr>
<tr>
<td>5.3. Character Set</td>
<td>8</td>
</tr>
<tr>
<td>6. TEXT SCREEN CONTROL CHARACTERS</td>
<td>9</td>
</tr>
<tr>
<td>6.1. Use of Teletext Colours</td>
<td>9</td>
</tr>
<tr>
<td>6.2. ASCII Control Codes</td>
<td>10</td>
</tr>
<tr>
<td>7. APPENDICES</td>
<td>11</td>
</tr>
<tr>
<td>7.1. Unpackaged Omegasoft Pascal</td>
<td>11</td>
</tr>
<tr>
<td>7.2. Using Software Interrupts in Pascal</td>
<td>14</td>
</tr>
<tr>
<td>7.3. Current Restrictions of Omegasoft Pascal Version 2.21</td>
<td>16</td>
</tr>
<tr>
<td>7.4. Input/Output Examples</td>
<td>17</td>
</tr>
</tbody>
</table>
1. **INTRODUCTION**

Omegasoft Pascal is supplied for POLY in two forms:

(i) A single-command method of compilation which is easy to use and requires the minimum of user interaction.

(ii) For users with a more extensive knowledge of the processes involved, there is a version which allows more flexibility. However, the compilation process is much more complex than with the single-command method. (See Appendix 7.1).

There is no difference in the Pascal language supported, only in the compilation process.

Omegasoft Pascal programs are written using the text editor resident on the POLY. The programs are then SAVEd on disk and the Pascal compiler executed.

Compilation of a Pascal program consists of three phases:

1. Compiling the Pascal source code into assembler code.

2. Converting the assembler code into 6809 relocatable machine language object code.

3. Linking the machine language program with the system subroutines used in the program.

As the Pascal program is compiled into 6809 machine language, the resulting compiled program executes extremely quickly.
2. PREPARING A PASCAL PROGRAM

The POLY Text Editor is described in the POLYSYS Utilities Manual. Basically the steps to use it are:

1. Load the text editor.
   This is done by entering
   
   TEXT
   from either DOS or BASIC
   
   Ready
   
   is displayed in CYAN.

2. Type in the Pascal program.
   The editing commands are described fully in the POLYSYS Utilities Manual. Note that the line numbers shown on the screen are for editing purposes only and are removed when the program is SAVED onto disk.

3. SAVE the program onto disk.
   Use the SAVE command. This saves the program as filename.TXT.
   
   For example:

   SAVE "MYPROG"

   saves your program as MYPROG.TXT.
The files supplied for the standard Pascal are:

- **PASCAL.CMD**: A DOS command which sets the standard options and controls the complete compilation process.
- **LL.CMD**: The linking loader.
- **PASLNK.RO**: A linking file.
- **PC.CMD**: The Pascal compiler.
- **RA.CMD**: The relocatable assembler.
- **RE.CMD**: The re-enter command.
- **RL.RO**: The subroutine library.

These files occupy approximately 340 sectors on disk and must be resident on the disk being used for the compilation.

To compile a Pascal program, enter DOS mode and run the Pascal compiler. The syntax of the command is:

```
PASCAL filename [S][D][L]
```

The filename is the name of the source file previously saved on disk. The default extension is .TXT. The resulting machine language program will be stored as a file with the name filename.CMD.

If S is specified then the compiler checks the program for syntax errors and no machine language program is produced.

If S is not specified and errors are found during the compile, then the machine language program is not created.

If L is specified, then a listing of the compilation on the printer is made.

If D is specified as well as L, then the listing is displayed on the screen, not the printer.

For example:

```
PASCAL MYPROG SL
```

compiles the source file MYPROG.TXT checking for syntax errors, producing a listing on the printer.

```
PASCAL MYPROG.TXT LD
```
compiles the source file MYPROG.TXT, displays a listing on the screen, and if no errors are found, creates the object program MYPROG.CMD.

Following compilation, the object program may be run by simply entering the filename.

For example:

    MYPROG

After a program has been run, it may be re-run by using the re-enter command.

For example:

    MYPROG

    :

    :

    RE

    will run MYPROG twice.
The following example Pascal program is supplied on disk in the file PRIMES.TXT:

```pascal
program fastprimes(INPUT,OUTPUT);
(* find the first 1229 primes *)
const n=1229; n1=35; (* sqrt of n *)
var i,k,x,inc,lim,square,lin: integer;
   prim: boolean;
   p,v: array(0..n) of integer;
begin
   writeln;
   write(2:8,3:8); lin:=2;
   x:=1; inc:=4; lim:=1; square:=9;
   for i:=3 to n do
   begin (* find next prime *)
      repeat x:=x+inc; inc:=6-inc;
          if square<=x then
          begin lim:=lim+1;
             v[lim.]:=square; square:=p[lim+1]*p[lim+1]
          end;
      k:=2; prim:=true;
      while prim and (k<lim) do
      begin k:=k+1;
         if v[k]<x then v[k.]:=v[k.]+2*p[k.]
         prim:=x<v[k.]
      end
      until prim;
      if i<=n1 then p[i.]:=x;
   write (x:8); lin:=lin+1;
   if lin=9 then
   begin writeln; lin:=0
   end
   writeln
end.
```

To edit the program, enter

TEXT,PRIMES

When complete enter

LIST

The file will be displayed on the POLY with line numbers. Move the cursor up to line 2, change 1229 to 1000 and press <ENTER>. Repeat for line 3. Then, enter

SAVE "NEWPRIME"
and the altered file will be saved as NEWPRIME.TXT.

Enter

PASCAL NEWPRIME L

When the compilation is complete

DOS

is displayed. Enter

NEWPRIME

to run the program.

Shift <EXIT> may be pressed at any time to terminate the program and return the POLY to DOS mode.
5. OMEGASOFT PASCAL ON POLY

5.1. Extensions to Omegasoft Pascal

(i) The device AUXOUT documented in the Omegasoft Pascal Manual as the printer, has not been implemented. To print on the printer, open a text file with .PRT as the extension. When the file is closed, it will be printed and deleted.

(ii) The Omegasoft Pascal compiler is intended to operate on single-user systems. Since a POLY network is a multi-user system, Pascal on POLY has been extended to provide for the locking and unlocking of random files, thus allowing more than one user to share the same random file. Two standard procedures, LOCK and UNLOCK, are provided so that when a random file is being accessed, the file may be locked to prevent access to spurious information. The syntax for calling these procedures is:

\[
\text{LOCK}(\text{device variable})
\]

\[
\text{UNLOCK}(\text{device variable})
\]

The LOCK statement locks the specified random file for 60 seconds. If the specified file is already locked, or if a SEEK is executed when the file has been locked by another user, runtime error 134 will be generated. Since the user will normally want to trap this error so as to retry the LOCK or SEEK operation until the file is unlocked, it is necessary to disable runtime I/O checking and use the function DEVERR. Following a successful seek operation, the file should be unlocked. An example using these procedures appears in Appendix 7.5.

(iii) When accessing an existing random file, it should be opened with the update option. The disk door should not be opened while the file is open, otherwise runtime error 132 (disk door opened while file open for write) will occur even if the file is being used for input only.

(iv) If runtime error checking is enabled and a runtime error occurs, the error will be printed with the corresponding line number in the Pascal source program. This facility has been added for ease of debugging. In order to minimise the code generated to support this facility, only executable Pascal lines (marked with an * on the listing) generate runtime line numbers. Thus if an "include file" (see page 1-14 of the Omegasoft Pascal manual) is specified, only one line number will be generated for the whole file. Because of this facility, the global and local stack frames in the runtime environment for POLY Pascal are 12 bytes long rather than 10 (see Appendix C of the Omegasoft Pascal manual). The additional two bytes are used for storing the source line number. This must be taken into account when writing procedures in assembler language (see the software interrupts example in section 7.2).
5.2. Conversion of Existing Programs

This is covered in Appendix F of the Omegasoft Pascal manual. If you have used assembler language procedures, the byte frame in the stack will have to be extended from 10 to 12 (as described in iv above).

5.3. Character Set

The POLY Teletext character set does not contain square or curly brackets and some characters have different representations. Pascal provides for this by accepting alternatives as follows. (*) and (*) may be used instead of curly brackets, and ( . and . ) instead of square brackets.

The Teletext hash (#) on POLY is the equivalent of the ASCII underline ( _ ), while the Teletext pound sign (#) is the equivalent of the ASCII hash (# ).

When writing comments curly brackets should be replaced with ( * and * ).

For example:

( * this is a comment * )

When using subscripts or sets the square brackets should be replaced with ( . or . ).

For example:

character := instr ing ( . 2 . )
digits := ( . '0' .. '9' . )

When declaring identifiers, the Teletext hash should be used in place of the ASCII underline.

For example:

var input # string : string;

When designating a byte constant, a Teletext pound sign should be used in place of the ASCII hash.

For example:

const enter = £ 13
6. TEXT SCREEN CONTROL CHARACTERS

6.1. Use of Teletext Colours

The screen is displayed using Teletext conventions whereas Pascal programs execute in ASCII mode. In order to use the Teletext control characters, a shift must be made into Teletext mode before using the Teletext control characters, and a shift back to ASCII after using them. The example below shows how the CHR function is used to incorporate these in an expression.

The teletext control characters are:

<table>
<thead>
<tr>
<th>ASCII DECIMAL VALUE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>Start RED characters</td>
</tr>
<tr>
<td>2</td>
<td>Start GREEN characters</td>
</tr>
<tr>
<td>3</td>
<td>Start YELLOW characters</td>
</tr>
<tr>
<td>4</td>
<td>Start BLUE characters</td>
</tr>
<tr>
<td>5</td>
<td>Start MAGENTA characters</td>
</tr>
<tr>
<td>6</td>
<td>Start CYAN characters</td>
</tr>
<tr>
<td>7 *</td>
<td>Start WHITE characters</td>
</tr>
<tr>
<td>8</td>
<td>Start FLASHING</td>
</tr>
<tr>
<td>9 *</td>
<td>End FLASHING</td>
</tr>
<tr>
<td>10</td>
<td>Not used</td>
</tr>
<tr>
<td>11</td>
<td>Not used</td>
</tr>
<tr>
<td>12 *</td>
<td>Normal height</td>
</tr>
<tr>
<td>13</td>
<td>Double height</td>
</tr>
<tr>
<td>14</td>
<td>Shift to ASCII mode</td>
</tr>
<tr>
<td>15</td>
<td>Shift to Teletext mode</td>
</tr>
<tr>
<td>16</td>
<td>Reverse video on</td>
</tr>
<tr>
<td>17</td>
<td>Start RED graphics</td>
</tr>
<tr>
<td>18</td>
<td>Start GREEN graphics</td>
</tr>
<tr>
<td>19</td>
<td>Start YELLOW graphics</td>
</tr>
<tr>
<td>20</td>
<td>Start BLUE graphics</td>
</tr>
<tr>
<td>21</td>
<td>Start MAGENTA graphics</td>
</tr>
<tr>
<td>22</td>
<td>Start CYAN graphics</td>
</tr>
<tr>
<td>23</td>
<td>Start WHITE graphics</td>
</tr>
<tr>
<td>24</td>
<td>CONCEAL display on rest of line</td>
</tr>
<tr>
<td>25 *</td>
<td>Contiguous graphics</td>
</tr>
<tr>
<td>26</td>
<td>Separated graphics</td>
</tr>
<tr>
<td>27</td>
<td>Reverse video off</td>
</tr>
<tr>
<td>28 *</td>
<td>No background to characters</td>
</tr>
<tr>
<td>29</td>
<td>Set background to current colour</td>
</tr>
<tr>
<td>30</td>
<td>Print graphics characters over</td>
</tr>
<tr>
<td></td>
<td>control characters</td>
</tr>
<tr>
<td>31 *</td>
<td>Print space for control characters</td>
</tr>
</tbody>
</table>
Each of the control characters occupies ONE screen position except the reverse video on character, the reverse video off character, the ASCII to Teletext shift character, and the Teletext to ASCII shift character. These characters do not require screen positions. All control characters are reset at the beginning of each line of the screen to those with an * beside them.

Double height characters extend down to the following line. If double height is used anywhere on a line, the following line is not displayed.

For example:

```pascal
program colour(output);
var
  num : integer;
  red,si,so : char;
begin
  red:= chr(1);
  si:= chr(14);
  so:= chr(15);
  for num:=1 to 100 do
    writeln(si,red,so,num);
end.
```

This prints out the numbers 1 to 100 in red. Note the shift to and from Teletext before and after the red colour control code.

6.2. ASCII Control Codes

<table>
<thead>
<tr>
<th>ASCII DECIMAL VALUE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Beep</td>
</tr>
<tr>
<td>8</td>
<td>Move the cursor 1 space to the left</td>
</tr>
<tr>
<td>9</td>
<td>Move the cursor 1 space to the right</td>
</tr>
<tr>
<td>10</td>
<td>Move the cursor down 1 line</td>
</tr>
<tr>
<td>11</td>
<td>Move the cursor up 1 line</td>
</tr>
<tr>
<td>12</td>
<td>Clear screen and move the cursor to the home position</td>
</tr>
<tr>
<td>13</td>
<td>Move cursor to the start of the line (RETURN)</td>
</tr>
<tr>
<td>14</td>
<td>Shift to ASCII mode</td>
</tr>
<tr>
<td>15</td>
<td>Shift to Teletext mode</td>
</tr>
<tr>
<td>16</td>
<td>Reverse video on</td>
</tr>
<tr>
<td>27</td>
<td>Reverse video off</td>
</tr>
<tr>
<td>30</td>
<td>Clear to the end of the line</td>
</tr>
</tbody>
</table>
7.1. Unpackaged Omegasoft Pascal

To use the unpackaged Pascal, an additional manual and further utilities are required.

The extra manual is:
- Omegasoft Relocatable Assembler and Linking Loader manual.

The extra utilities are:

CHAIN.CMD The linkage creation control program.
DB.CMD The debugger.
LB.CMD Librarian for the runtime routines.
LC.CMD The linkage-file creator.

To compile a Pascal program use the following steps:
1. Run the Pascal Compiler.
2. Run the Linkage Creator.
3. Run the Assembler and Linking Loader.

The syntax for using the Pascal compiler for unpackaged use is:

```
PC <source-file [>output-file 0] [>>print-file L]
```

Source-file is the file containing the Pascal program, the default extension is .TXT. The output-file is the compiler output file for the next phase of the compilation, the default extension being .CO. The print-file is the compiler listing.

The syntax for using the Linkage Creator for unpackaged use is:

```
LC
```

The Linkage Creator prompts the user for various parameters for the assembling and linking of the program, and produces a command file (extension .CF) and a pre-setup source file (extension .PS) for use in the next step.
The Assembler and Linking Loader steps are combined in the command file output by the Linkage Creator which is invoked by the CHAIN command. The syntax for CHAIN

CHAIN filename

where filename is the output file from the linkage creator (default extension .CF). This will cause the compiler output file (extension .CO) to be assembled producing a relocatable object file (extension .CA). As well, the pre-setup source file (extension .PS) produced by the Linkage Creator will be assembled producing another relocatable object file (extension .PA). The Linking Loader then links these relocatable object files producing a binary object file (extension .BIN).

Shown below is an example showing the steps used to compile PRIMES using the unpackaged method (user input is shown underlined).

PC <PRIMES >PRIMES 0 >>PRIMES L
Omegasoft 6809 Pascal Compiler Version 2.21
Copyright 1983 by Certified Software Corporation

LC
Omegasoft 6809 Linkage Creator Version 2.20
Copyright 1983 by Certified Software Corporation
Pascal compiler output file name : PRIMES
Pascal program name : FASTPRIMES
Auto setup ? Y
System stack size : 512
Starting load location : 100
Library drive number : <ENTER> (current drive used)
Additional files to load : <ENTER> (no extra files)
Additional library files : <ENTER> (no extra files)
Load options : <ENTER> (no Load options)
Map options : <ENTER> (no map options)

CHAIN PRIMES
Omegasoft 6809 Chain version 2.10
Copyright 1983 by Certified Software Corporation
RA <PRIMES.CO >PRIMES.CA 0
Omegasoft 6809 Relocatable Assembler version 1.40
Copyright 1983 by Certified Software Corporation
Total errors : 0 Psct size : 0292 Table usage : 35 of 1949
RA <PRIMES.PS >PRIMES.PA 0
Omegasoft 6809 Relocatable Assembler version 1.40
Copyright 1983 by Certified Software Corporation
Total errors : 0 Psct size : 000F Table usage : 4 of 1949
LL
Linking Loader version 1.40
Copyright 1983 by Certified Software Corporation
?STRP=$0100
?LOAD=PRIMES.PA PRIMES.CA
?LIB= RL
?OBJA=PRIMES.BIN
?STRP=S0100
?LOAD=PRIMES.PA PRIMES.CA
?LIB= RL
?OBJA=PRIMES.BIN
?MAPC
PSCT SIZE=09A7 START=0100 END=0AA6
SYMBOL TABLE USAGE: USED=50 of 1930
MODULE TABLE USAGE: USED=17 of 79
?EXIT

End chain

The above example will result in a file PRIMES.BIN, which will be identical with PRIMES.CMD produced by the command:

PASCAL PRIMES L
7.2. Using Software Interrupts in Pascal

Pascal has no command that can directly call software interrupts. Software
interrupts must be called from assembler code which may be directly
imbedded in a Pascal program.

For example:

program swi#example(input,output);
(* example program using software
  interrupts duplicates the first
  example for SPLIT in the POLYBASIC
  Manual *)
procedure cursor(row,column:integer);
(* position the cursor at specified text
  row and column. If the values
  supplied are outside the screen
  then the cursor is not moved *)
begin
  PSHS D,X,Y,U       Save registers
  LEAU 12,U          Skip over the linkage data
  LEAU 1,U           Point to low order byte of column
  LDB 0,U++          Load B with the low order byte of column
  LDA 0,U            Load A with the low order byte of row
  SWI
  FCB 9              Position cursor
  PULS D,X,Y,U       Restore the registers
end;
procedure split(splitat:integer);
(* Split the screen into two separate
  scrolling sections. The parameter
  is the row at which the split is
  to be done. If the parameter is
  0 or >24 then split is turned off *)
begin
  PSHS D,X,Y,U       Save the registers
  LEAU 12,U          Skip over the linkage data
  LDD 0,U            Load D with the 16 bit split-row value
  EXG A,B            Swap byte positions of D
  SWI                Split screen
  FCB 13             Restore the registers
end;
procedure wait(milliseconds:integer);
(* wait for the specified number of
  milliseconds *)
begin
  PSHS D,X,Y,U       Save the registers
  LEAU 12,U          Skip over the linkage data
  LDD 0,U            Load the time to wait
  SWI                Wait for x milliseconds
  FCB 19             Restore the registers
end;
procedure numberprint;

var
  loop: integer;

begin
  for loop:=1 to 21 do
    writeln(loop:2);
    wait(20);
end;

begin (* mainline *)
  page;
  split(10);
  numberprint;
  cursor(10,0);
  numberprint;
  wait(500);
  split(0);
end.
7.3. Current Restrictions of Omegasoft Pascal
Version 2.21

Omegasoft has documented the following restriction:

Structures may not include devices as part of the structure. This includes arrays of devices or records containing devices. If this type of structure is desired it is possible to define the array or record as containing pointers to a device and using the addr function to set the pointer.
7.4. Input/Output Examples

Listed below is a simple example illustrating techniques for disk input/output using Pascal on POLY.

```
program inputoutput(input,output);
var
    eofflag : boolean;
    filename,buffer : string;
    file1 : text;
begin
    eofflag:=false;
    writeln('Enter file name: ');
    readln(filename);
    (* open the file for output *)
    rewrite(file1,filename);
    (* put data into the file *)
    repeat
        readln(buffer);
        (* if the first character of the
           input string is a "#" then input
           is complete *)
        if buffer (.1)='#' then eofflag:=true
        else
            writeln(file1,buffer);
    until eofflag;
    (* data input complete, close file *)
    close(file1);
    (* open the new file for input *)
    reset(file1,filename);
    (* read the data back from the
       file and display it on screen *)
    while not eof(file1) do
        begin
            readln(file1,buffer);
            writeln(buffer);
        end;
end.
```
Listed below is a further example illustrating techniques for using random files with Pascal on POLY.

program Randomfile(input,output);
label 11;
const
  lock_error = #134;
type
  rekord = string(.62.);
var
  eofflag : boolean;
  rec, filename, buffer : rekord;
  filel : file of rekord;
  I : integer;
begi
  eofflag := false;
  write('Enter file name: ');
  readln(filename);
  (* open the file for output *)
  create(filel, filename, output, 1);
  (* put data into the file *)
  repeat
    readln(buffer);
    (* if the first character of the 
      input string is a "#" then input 
      is complete *)
    if buffer(.1.) = ' #' then eofflag := true
    else
      begin
        filel^ := buffer;
        put(filel);
      end;
  until eofflag;
  (* data input complete, close file *)
  close(filel);
  (* open the file for input *)
  open(filel, filename, update, 1);
  (* read the data back from the 
     file and display it on screen *)
  I := 0;
  while I >= 0 do
    begin
      write('Which record do you want to fetch? ');
      readln(I);
      if I >= 0 then
        begin
          (*$I- turn I/O error check off *)
        11: lock(filel);
        if DEVERR(filel) <> #0 then
          if DEVERR(filel) = lock_error then goto 11
          else halt(DEVERR(filel));
          (*$I+ turn I/O error check on *)
        seek(filel, I);
        unlock(filel);
      end;
    end;
buffer := file1^;
writeln(buffer);
end;
end;
close(file1);
end.

Two warnings when using random files. Since Pascal blocks logical records into physical disk sectors, the last sector written may contain logical records not written but which are present to fill out the sector. These records may be read without causing a non-existent record number runtime error. The user is thus advised to keep track of the number of logical records actually written, storing this, for example, in the first record of the random file. Note also that Pascal commences to number records of a random file with 0.

Further, if the last logical record written actually coincides with the end of a physical disk sector, then it will not be possible to reference this record (as with buffer := file1^ in the above example). This is because Pascal employs a read-ahead method of input. It will successfully seek to the last logical record and commence to retrieve the individual bytes. However, when the last byte is being retrieved, the read-ahead mechanism will cause an attempt to read the next disk sector resulting in a non-existent record number error. This may occur if any multiple of your record length plus 1 is also a multiple of 252. If this is the case, then write a dummy record at the end of your file. Note that the actual length of a random record is one greater than the declared length. The extra byte is used by Pascal to store the length of valid data in the record. Thus, in the above example, the declared record length is 62 and the actual record length is 63. This divides exactly into 252 so there will be 4 records per physical disk sector, and it will not be possible to retrieve and print the last record stored if this last record is record number 3, 7, 11, etc., since any of these records will coincide with the end of a physical disk sector.