G-PASCAL NEWS

Registered by Australia Post - Publication Number VBG 6589

Published by Gambit Games for Commodore 64 G-Pascal owners.

Mail: G-Pascal News, P.O. Box 124, Ivanhoe, Victoria 3079. (Australia)
Phone: (03) 497 1283

Gambit Games is a trading name of Gammon & Gobbett Computer Services Proprietary Limited, a company incorporated in the State of Victoria.

VOLUME 2, NUMBER 1 - January 1985 Annual Subscription $12

WHAT'S IN THIS ISSUE

This issue contains quite a few interesting and useful programs in G-Pascal:

° A name and address filing program for disk drives - this program illustrates how to access random disk files.

° A "pretty-print" program for G-Pascal source programs - this program will take any program and properly indent it.

° An updated version of MODEM64 - this version is now over 1100 lines long and has many useful additions - it can even be used as a file utility (examine and copy disk files) if you don't have a modem.

° Instructions on how to back up your G-Pascal compiler.

° How to compile programs over 16K in size.

° How to automatically default to loading from disk or cassette.

TYPING IN PROGRAMS

All of the programs in this issue have been printed on a daisy-wheel printer direct from G-Pascal programs on disk. They should all work properly if keyed in exactly as shown. Please note that the special printing program we use shows reserved words in upper case, to aid in reading and understanding programs. It is not necesssary to key reserved words (such as BEGIN, END) in upper case - in any event they will be displayed in lower case on your screen. Also, be careful with the following characters:

1 Number one.
l Lower case letter 'L'.
I Upper case letter 'I'.
Ø Number zero.
0 Upper case letter 'O'.
[Press SHIFT and :
] Press SHIFT and ;
(Press SHIFT and 8
) Press SHIFT and 9
_ Underscore (above CTRL key).

BACKING UP YOUR G-PASCAL

To back up your copy of G-Pascal, load G-Pascal as usual, make any 'patches' that you normally make (such as changing the printer secondary address, or the screen colour), and then enter, compile and run the following program:

```
1 BEGIN
2 SAVE (8, $8ØØØ, $bfff, "gpascal backup")
3 END .
```

If you want to save to cassette rather than disk then the first '8' on line 2 should be a '1'.

When you run the program it will create a file called "gpascal backup" on your disk or cassette. This is your backup copy. As this copy does not have any 'auto-load' program at the front of it you will have to load it slightly differently from the usual method. To load and run the backup copy (from Basic, after turning on the power to the C64) enter:

```
LOAD "GPASCAL BACKUP",8,1
SYS 32768
```

Once again, if you are loading from cassette the '8' will be a '1'.

This information is given so you can make backup copies for your own personal use - please do not give away (or sell) copies to other people. The G-Pascal compiler is copyright.

YOUR PROGRAMS SOLICITED

If you have developed a nifty program or subroutine in G-Pascal that you would like to share, please send it in to us. We are also interested in hearing about programs written in G-Pascal that are suitable for sale.

MODEM64 - VERSION 1.6

This previously unpublished listing of MODEM64 features a number of enhancements to earlier versions. In particular it now has the capability to:

° Optionally save conversations in memory.

° Display a previous conversation or file on the screen.

° Print a previous conversation or file on the printer.

° Save a conversation to disk or cassette.

° Analyse the current status (length etc.) of the current contents of the memory buffer.

° Loading and transmitting files are now separate functions, so you can load a file, display its contents and then optionally transmit it to the other end.

° The program checks whether you want to go ahead before allowing you to do anything which will clobber any existing contents of memory (i.e. quit, load or receive).

° The file reception code has been tidied up slightly to improve its response to timeout situations.

° Verification of saves to disk or cassette is now optional.

° It is possible to display on the screen each block of a file that is being transmitted or received during transmission.

° Certain aspects are more user-friendly.

Entering MODEM64

You will need to key in the program from the listing. If you have an earlier version you can save a lot of effort by loading your version and (using the editor List command) compare it to this listing, altering, deleting or adding lines where necessary. Please note that some comments, particularly the lines of asterisks below procedure declarations, have been removed so that the program will fit into 16K. Also, some of the changes between versions are purely 'cosmetic', so that the listing is reasonably narrow for inclusion in this magazine - for example, splitting a statement over two or more lines. Such changes are optional, of course. You may wish to make them in any case so that your line numbers agree with our listing.

You can 'tailor' the program to your requirements by changing the constants at lines 19 to 24. For example, you can change the printer channel and secondary address. If you make 'display_file' true then each block of data will be displayed on the screen during transmission and reception. By changing 'max_retries' you can alter the program's tolerance to timeouts etc. If you make 'verify_wanted' false then saves to disk or cassette will not be verified. If you make 'receive_with_crck' false then each block of data during reception will only have a simple sum check rather than a cyclic redundancy check. This is not recommended in general. (During transmission this option is controlled by the other end.)

If you are using a printer you may also need to add extra code at lines 983-984, instead of the comments that are there now, depending on your printer type.

USING MODEM64 Version 1.6

First load, compile and run MODEM64. The program automatically enters full duplex terminal mode, ready for communicating with a bulletin board or another computer (you can change this by changing line 174).

To select the main menu just press the Commodore key on its own.

To converse with another Commodore 64 owner you will need to select half-duplex mode ('H') so that you can see what you are typing.

Conversations are automatically saved in memory (and appended to a previous conversation or file). To toggle capture of conversations into memory press 'M' at the main menu - this will toggle the current memory-capture state, and inform you of what the new one is. To erase the last conversation and start afresh, enter 'E' - you will be told the size of the conversation and asked to confirm the erasure (unless there is nothing there).

For any option that requests a file name you can now press RETURN on its own to escape from that option (without loading or saving the file).

To send a file, first load it into memory if necessary, using the 'L' (Load) option. You will be told how big it is and how long (roughly) it will take to transmit. At this stage it would be wise to make sure that conversation capture is off, so that anything you type now does not get appended to the file - use the 'M' option described above if necessary. Then synchronise with the other computer (by voice or by typing a message). When the other end is ready to receive press 'S' (Send file). You should see an asterisk as each block is transmitted, or if you have selected the 'display' option then you will see the contents of the block itself. Initial synchronisation may take a minute, so be patient. After the file is transmitted (you will get a message to this effect) you are automatically returned to terminal mode.

As each block of data has a cyclic redundancy check (CRCK) appended to it the likelihood of a mis-transmission is very low. You can further check by manually comparing the 'file' CRCK figure which is

displayed when the file is transmitted or received, to the figure that the person at the other end got.

To see what the current status of the contents of the memory buffer is, press 'A' - Analyse Memory.

To type the contents of the memory buffer, press 'T' - then press SHIFT to temporarily halt the display (press SHIFT/LOCK for a longer pause). To abort the listing press the Commodore key.

To print the contents of memory select 'P' - this works the same as 'T' except that output is directed to your printer.

To receive a file, just synchronise with the other end - when they are ready to transmit just press 'R' for Receive. You will see an asterisk as each block is received, or an error message after 10 seconds if nothing is happening. When the file has been received you will be asked for the name to save it under. You can just press RETURN at this stage if you like, and save it later using the 'D' option.

The 'D' option (Dump conversation to disk/cassette) saves the current contents of the memory buffer (conversation or file) to disk or cassette. You can actually use MODEM64 as a file copying utility by just loading a file (using 'L') and then saving it to another disk using 'D'. You can also use it to download programs or data from another computer that doesn't support the Christensen protocol. Just save the conversation in memory and dump it to disk at the end.

The 'C' (Cancel transmission) option is for aborting a transmission currently in progress. In this case you will need to first abort your end of the transmission by pressing RUN/STOP. This will take you to the G-Pascal Main Menu. Re-run MODEM64 by pressing 'R'. Then press the Commodore key to get the MODEM64 menu. Then select 'C' - this will transmit three 'CAN' characters (hex 18) to the other computer, telling it to cancel sending or receiving the file. This option would not normally be used, but is included as a means from escaping from a long transmission started in error - the wrong file name perhaps?

Another enhancement to this version is the provision of a 'break' capability. Transmitting a break involves sending at least 10 consecutive zero bits to the other computer.

To transmit a break with MODEM64 just press [f1] - that is, function key 'f1' on the right hand side of the keyboard. The 'break' is implemented by closing the RS232 file, bringing the output port to zero for 1/10th of a second, and re-opening the file - see lines 937 to 944.

```
1 (* MODEM64  - January 1985 Version
2
3    Author: Nick Gammon.
4    Public Domain Program.
5
6    %a $840  (P-codes start at $840)
7 *)
8
9 CONST
10       bs = 8;
11       ff = 12;
12       cr = 13;
13       fs = 28;
14       ctrlz = $1a;
15       home = 147;
16       true = 1;
17       false = 0;
18
19       printer_channel = 4;
20       printer_sec_addr = 0;
21       display_file = false;
22       receive_with_crck = true;
23       max_retries = 6;
24       verify_wanted = true;
25
26       charcolour = 10;
27       white = 1;
28       green = 5;
29       light_red = 10;
30       light_green = 13;
31       light_blue = 14;
32       light_grey = 15;
33
34       start_address = $2100;
35       cassette = 1;
36       disk = 8;
37       areg = $2b2;
38       xreg = $2b3;
39       yreg = $2b4;
40       cc = $2b1;
41       setlfs = $ffba;
42       setnam = $ffbd;
43
44       soh = $1;
45       eot = $4;
46       ack = $6;
47       nak = $15;
48       can = $18;
49       rs232_status = $297;
50       empty = 8;
51
52 VAR
53     command : CHAR ;
54
55     buffer : ARRAY [130] OF CHAR ;
56     name1, name2 : ARRAY [20] OF CHAR ;
57     last_terminal_mode,
58     medium,
59     got_medium,
60     length,
61     bad_result,
62     next_address,
63     final_address,
64     retries,
65     capture,
66     eof,
67     abort,
68     bad_block,
69     seq_error,
70     bad_sum_check,
71     timeout,
72     block_no,
73     inverse_block_no,
74     expected_block,
75     last_block,
76     want_crck,
77     sum_check_received,
78     sum_check_received_2,
79     sum_check,
80     sum_check_2  : INTEGER ;
81     routine : ARRAY [35] OF INTEGER ;
82
83 FUNCTION commodore_logo;
84 BEGIN
85   commodore_logo :=
86         MEMC [653] AND 2 <> 0
87 END ;
```

```
88
89 FUNCTION shift_key_pressed;
90 BEGIN
91   shift_key_pressed :=
92         MEMC [653] AND 1 <> 0
93 END ;
94
95 PROCEDURE ink (colour);
96 BEGIN
97   GRAPHICS (charcolour, colour)
98 END ;
99
100 PROCEDURE open_rs232_file;
101 CONST
102       openit = $ffc0;
103 VAR name : ARRAY [1] OF CHAR ;
104 BEGIN
105 (* first set up the file name
106    as per the RS232 paramters *)
107
108   name [1] := 6;  (*  300 baud  *)
109   name [0] := 0;    (*  3-line    *)
110   MEMC [$f8] := $c1; (* buffer *)
111   MEMC [$fa] := $c2; (* buffer *)
112   MEMC [areg] := 2;
113   MEMC [xreg] := 2; (* RS232 *)
114   MEMC [yreg] := 2;
115   CALL (setlfs);
116   MEMC [areg] := 2;
117   MEMC [xreg] := ADDRESS (name[1]);
118   MEMC [yreg] := ADDRESS (name[1]) SHR 8;
119   CALL (setnam);
120   CALL (openit)
121 END ;
122
123 PROCEDURE init;
124
125 CONST colour = 1;
126       point = 2;
127       behindbk = 6;
128
129 VAR i : INTEGER ;
130
131 PROCEDURE insert(x, y, z);
132 BEGIN
133   routine [i] := x;
134   routine [i - 1] := y;
135   routine [i - 2] := z;
136   i := i - 3
137 END ;
138
139 BEGIN (* init *)
140   WRITE (CHR (home));
141   ink (light_grey);
142   MEMC [650] := 128;
143   (* all keys auto-repeat *)
144   WRITELN
145   ("YAM-compatible Modem Program for C64.");
146   WRITELN
147   ("Written by Nick Gammon in G-Pascal.");
148   WRITELN
149   ("Version 1.6  - PUBLIC DOMAIN.");
150   WRITELN
151   ("G-Pascal is produced by Gambit Games -");
152   WRITELN
153   (" enquiries: Gambit Games, P.O. Box 124,");
154   WRITELN
155   (" Ivanhoe, Victoria 3079. Australia.");
156   WRITELN ;
157   i := 35;
158   (* crck routine for transmission *)
159   insert($8500a9,$5f855e,$854bb1);
160   insert($08a207,$260726,$5f265e);
161   insert($a50c90,$10495f,$a55f85);
162   insert($21495e,$ca5e85,$88e9d0);
163   insert($60e0d0,0,0);
164   (* crck routine for file *)
165   insert($8500a9,$068505,$0506a8);
166   insert($080626,$184bb1,$850565);
167   insert($902805,$97490a,$a50585);
168   insert($a04906,$e60685,$02d04b);
169   insert($a54ce6,$5ec54b,$a5dbd0);
170   insert($5fc54c,$a5d5d0,$4b8505);
171   insert($8506a5,$ff604c,0);
172   buffer [128] := 0;
173   buffer [129] := 0;
174   command := "f";
175   DEFINESPRITE (32,
176    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff);
177   SPRITE (1, point, 32,
178           1, colour, light_grey,
179           1, behindbk, true);
180   got_medium := false;
181   capture := true;
182   final_address := start_address;
183   open_rs232_file
184 END ; (* of init *)
185
186 PROCEDURE start_error;
187
188 BEGIN
189   ink (light_red);
190   WRITELN
191 END ;
192
193 PROCEDURE error;
194
195 BEGIN
196 IF expected_block <> -1 THEN
197    WRITE (" on block ",
198              expected_block)
199 ELSE
200    WRITE (" on EOT");
201 WRITELN (" retry ", retries);
202 retries := retries + 1;
203 ink (green);
204 IF retries > max_retries THEN
205   abort := true
206 END ;
207
208 PROCEDURE get_file_name;
209 VAR i, got_cr : INTEGER ;
210     ch : CHAR ;
211 BEGIN
212 IF NOT got_medium THEN
213   BEGIN
214   WRITELN ;
215   WRITE ("<D>isk or <C>assette? ");
216   ink (light_blue);
217   REPEAT
218     READ (ch);
219     ch := ch AND $7f
220   UNTIL (ch = "d")
221      OR (ch = "c");
222   WRITELN (CHR (ch));
223   ink (green);
224   IF ch = "d" THEN
225     BEGIN
226     medium := disk;
227     OPEN (15, disk, 15, "i")
228     END
229   ELSE
230     medium := cassette;
231   got_medium := true
232   END ;
233 WRITELN ;
234 WRITE ("File name? ");
235 ink (light_blue);
236 READ (name1);
237 ink (green);
238 got_cr := false;
239 FOR i := 0 TO 20 DO
240   IF NOT got_cr THEN
241     BEGIN
242     name2 [20 - i] := name1 [i];
243     IF name1 [i] = cr THEN
244       BEGIN
245       length := i;
246       got_cr := true
247       END
248     END
249 END ;
250
251 PROCEDURE check_result;
252
253 CONST readst = $ffb7;
254
255 VAR i, error_code : INTEGER ;
256     result : ARRAY [80] OF CHAR ;
257 BEGIN
258   IF MEMC [cc] AND 1 THEN
259     error_code := MEMC [areg]
260   ELSE
261     BEGIN
```

```
262     CALL (readst);
263     error_code := MEMC [areg] AND $bf
264     END ;
265   bad_result := error_code;
266   IF medium = disk THEN
267     BEGIN
268     GET (15);
269     READ (result);
270     GET (0);
271     result [80] := cr;
272     IF (result [0] <> "0")
273     OR (result [1] <> "0") THEN
274       BEGIN
275       bad result := true;
276       i := -1;
277       start_error;
278       REPEAT
279         i := i + 1;
280         WRITE (CHR (result [i]))
281       UNTIL result [i] = cr
282       END
283     END ;
284   WRITELN ;
285   IF error_code THEN
286     BEGIN
287     start_error;
288     WRITELN ("File error, code: ",
289             error_code)
290     END ;
291   ink (green);
292   IF NOT bad_result THEN
293     WRITELN ("Ok.")
294 END ;
295
296 PROCEDURE load_nominated_file (flag);
297
298
299 PROCEDURE load_file;
300 CONST
301       loadit = $ffd5;
302
303 BEGIN
304   MEMC [areg] := 1;
305   MEMC [xreg] := medium;
306   MEMC [yreg] := 0; (* relocate *)
307   CALL (setlfs);
308   MEMC [areg] := length;
309   MEMC [xreg] := ADDRESS (name2[20]);
310   MEMC [yreg] := ADDRESS (name2[20]) SHR 8;
311   CALL (setnam);
312   MEMC [areg] := flag;  (* load /verify *)
313   MEMC [xreg] := start_address;
314   MEMC [yreg] := start_address SHR 8;
315   CALL (loadit);
316   check_result
317 END ;
318
319 BEGIN
320 REPEAT
321   IF flag = 0 THEN  (* load *)
322     get_file_name;
323   IF length <> 0 THEN
324     load_file
325 UNTIL (bad_result = 0)
326    OR (flag = 1)
327    OR (length = 0)
328 END ;
329
330 PROCEDURE save_nominated_file;
331
332
333 PROCEDURE save_file;
334
335 CONST saveit = $ffd8;
336       register = $6a;
337 BEGIN
338   MEMC [areg] := 1;  (* file no *)
339   MEMC [xreg] := medium;
340   MEMC [yreg] := 0;
341   CALL (setlfs);
342   MEMC [areg] := length;
343   MEMC [xreg] :=
344         ADDRESS (name2 [20]);
345   MEMC [yreg] :=
346         ADDRESS (name2 [20]) SHR 8;
347   CALL (setnam);
348   MEMC [register] := start_address;
349   MEMC [register + 1] :=
350               start_address SHR 8;
351   MEMC [areg] := register;
352   MEMC [xreg] := final_address;
353   MEMC [yreg] := final_address SHR 8;
354   CALL (saveit);
355   check_result
356 END ;
357
358 BEGIN
359 REPEAT
360   get_file_name;
361   IF length <> 0 THEN
362     BEGIN
363     save_file;
364     IF (NOT bad_result)
365     AND verify_wanted THEN
366       BEGIN
367       IF medium = cassette THEN
368         BEGIN
369         WRITELN ;
370         WRITELN
371    ("Rewind cassette to save point for");
372         WRITE
373    ("verification - press <SHIFT>");
374         WRITELN (" when ready.");
375         REPEAT UNTIL shift_key_pressed
376         END ;
377       load_nominated_file (1) (* verify *)
378       END
379     END
380 UNTIL (NOT bad_result) OR (length = 0)
381 END ;
382
383 FUNCTION from_modem;
384
385 BEGIN
386   GET (2);
387   from_modem := GETKEY ;
388   GET (0)
389 END ;
390
391 PROCEDURE display_char (x);
392
393 BEGIN
394   x := x AND $7f;
395
396 (* Reverse upper/lower case *)
397
398   IF (x >= $61) AND
399      (x <= $7a) THEN
400      x := x - $20
401   ELSE
402   IF (x >= "a") AND
403      (x <= "z") THEN
404      x := x + $80;
405
406 (* Only display if printable *)
407
408   IF (x >= " ")
409   OR (x = cr) THEN
410      WRITE (CHR (x))
411   ELSE
412     IF x = bs THEN
413       WRITE (CHR (157))
414     ELSE
415       IF x = fs THEN
416         WRITE (CHR (29))
417       ELSE
418         IF x = ff THEN
419           WRITE (CHR (home))
420 END ;
421
422 PROCEDURE to_modem (x);
423
424 BEGIN
425   PUT (2);
426   WRITE (CHR (x));
427   PUT (0)
428 END ;
429
430 FUNCTION calc_crck;
431
432 BEGIN
433   MEMC [$4b] := ADDRESS (buffer [130]);
434   MEMC [$4c] := ADDRESS (buffer [130]) SHR 8;
435   MEMC [yreg] := 130;
```

```
436   CALL (ADDRESS (routine[35]));
437   calc_crck := MEM [$5e] AND $ffff
438 END ;
439
440 PROCEDURE calc_file_crck;
441
442 BEGIN
443   MEMC [$4b] := start_address;
444   MEMC [$4c] := start_address SHR 8;
445   MEMC [$5e] := final_address;
446   MEMC [$5f] := final_address SHR 8;
447   CALL (ADDRESS (routine[20]));
448   WRITELN ("Cyclic redundancy check = $",
449            HEX (MEM [$4b] AND $ffff))
450 END ;
451
452 FUNCTION next_char (period);
453
454 CONST count_per_second = 145;
455 VAR ch : CHAR ;
456     counter : INTEGER ;
457 BEGIN
458 counter := period * count_per_second;
459 REPEAT
460   ch := from_modem;
461   counter := counter - 1
462 UNTIL (NOT (MEMC [rs232_status] AND empty))
463    OR (counter <= 0);
464 timeout := MEMC [rs232_status] AND empty <> 0;
465 next_char := ch
466 END ;
467
468 PROCEDURE purge;
469
470 VAR discard : CHAR ;
471 BEGIN
472 REPEAT
473   discard := next_char (1)
474 UNTIL timeout
475 END ;
476
477 PROCEDURE send_nak;
478
479 BEGIN
480   purge;
481   IF (expected_block = 1)
482   AND want_crck THEN
483     to_modem ("c")
484   ELSE
485     to_modem (nak)
486 END ;
487
488 PROCEDURE cancel_trans;
489
490 BEGIN
491   purge;
492   to_modem (can);
493   to_modem (can);
494   to_modem (can);
495   start_error;
496   WRITELN ("Transmission aborted")
497 END ;
498
499 PROCEDURE receive_block;
500
501 VAR ch : CHAR ;
502     i : INTEGER ;
503 BEGIN
504 bad_block := false;
505 block_no := next_char (1);
506 IF NOT timeout THEN
507   inverse_block_no := next_char (1);
508 IF NOT timeout THEN
509 IF (block_no + inverse_block_no + 1)
510    AND $ff <> 0 THEN
511   BEGIN
512   start_error;
513   WRITE ("Bad block no.");
514   error;
515   send_nak;
516   bad_block := true
517   END
518 ELSE
519   IF ((block_no = last_block AND $ff)
520   AND (expected_block <> 1))
521   OR (block_no = expected_block AND $ff) THEN
522     seq_error := false
523   ELSE
524     BEGIN
525     seq_error := true;
526     start_error;
527     WRITELN ("Block number sequence error");
528     error;
529     send_nak
530     END ;
531 IF NOT (bad_block OR seq_error) THEN
532   BEGIN
533   sum_check := 0;
534   FOR i := 0 TO 127 DO
535     IF NOT timeout THEN
536       BEGIN
537       ch := next_char (1);
538       buffer [i] := ch;
539       sum_check := sum_check + ch
540       END ;
541   IF NOT timeout THEN
542     sum_check_received := next_char (1);
543   IF want_crck THEN
544   IF NOT timeout THEN
545     sum_check_received_2 := next_char (1)
546   END ;
547 IF timeout THEN
548 BEGIN
549   start_error;
550   WRITE ("Timeout on receive");
551   error;
552   send_nak
553   END
554 ELSE
555 IF NOT (bad_block OR seq_error) THEN
556   BEGIN
557   bad_sum_check := true;
558   IF want_crck THEN
559     IF calc_crck = sum_check_received SHL 8
560     OR sum_check_received_2 THEN
561       bad_sum_check := false
562     ELSE
563   ELSE
564     IF sum_check AND $ff =
565         sum_check_received THEN
566       bad_sum_check := false;
567   IF bad_sum_check THEN
568     BEGIN
569     start_error;
570     WRITE ("Sum check error");
571     error;
572     send_nak
573     END
574   ELSE
575     BEGIN
576     to_modem (ack);
577     retries := 0;
578     IF block_no = expected_block AND $ff THEN
579       BEGIN
580       last_block := expected_block;
581       expected_block := expected_block + 1;
582       IF display_file THEN
583           FOR i := 0 TO 127 DO
584             display_char (buffer [i])
585       ELSE
586           WRITE ("*");
587       FOR i := 0 TO 127 DO
588           BEGIN
589           MEMC [next_address] :=
590               buffer [i];
591           next_address := next_address + 1
592           END
593       END
594     END
595   END
596 END ;
597
598 PROCEDURE receive_block_can_eot;
599
600 VAR ch : CHAR ;
601 BEGIN
602 REPEAT
603   ch := next_char (10)
604 UNTIL (ch = soh)
605    OR (ch = eot)
606    OR (ch = can)
607    OR timeout;
608 IF timeout THEN
609   BEGIN
610   start_error;
611   WRITE ("Timeout at start");
```

```
612   error;
613   send_nak
614   END
615 ELSE
616   CASE ch OF
617     soh: receive_block;
618     can: BEGIN
619          start_error;
620          WRITELN ("Sender CANcelled transmission");
621          abort := true
622          END ;
623     eot: BEGIN
624          eof := true;
625          to_modem (ack)
626          END
627   END   (* of case *)
628 END ;
629
630 PROCEDURE receive_file;
631
632 BEGIN
633 WRITELN ;
634 ink (light_green);
635 WRITELN ("[Receive a File]");
636 ink (green);
637 WRITELN ;
638 expected_block := 1;
639 last_block := 0;
640 retries := 0;
641 abort := false;
642 eof := false;
643 seq_error := false;
644 next_address := start_address;
645 want_crck := receive_with_crck;
646 send_nak;  (* get things going *)
647 REPEAT
648   receive_block_can_eot
649 UNTIL abort OR eof OR seq_error;
650 WRITELN ;
651 IF eof THEN
652   BEGIN
653   final_address := next_address;
654   WRITELN ;
655   WRITELN ("File received successfully");
656   calc_file_crck;
657   save_nominated_file
658   END
659 ELSE
660   BEGIN
661   final_address := start_address;
662   cancel_trans  (* stop other end *)
663   END
664 END ;
665
666 PROCEDURE analyse_file (loadit);
667
668 VAR
669     file_length, blocks, mins : INTEGER ;
670 BEGIN
671 WRITELN ;
672 IF loadit THEN
673   BEGIN
674   load_nominated_file (0);
675   IF length <> 0 THEN
676     BEGIN
677     final_address := MEMC [xreg] + MEMC [yreg] SHL 8;
678     file_length := final_address - start_address;
679     WHILE file_length AND $7f <> 0 DO
680       BEGIN
681       file_length := file_length + 1;
682       MEMC [final_address] := ctrlz;
683       final_address := final_address + 1
684       END
685     END
686   END ;
687 blocks := (final_address -
688    start_address + 127) / 128;
689 IF blocks <> 0 THEN
690   BEGIN
691   mins := blocks * 561 / 600;
692   WRITELN (blocks, " blocks,  ",
693            blocks * 10 / 80,
694            ".",
695            blocks * 10 / 8 MOD 10,
696            " K");
697   calc_file_crck;
698   WRITELN ("Transmission time: ",
699            mins / 10, ".",
700            mins MOD 10,
701            " minutes.")
702   END
703 END ;
704
705 PROCEDURE process_can;
706
707 BEGIN
708   start_error;
709   WRITELN ("Receiver CANcelled transmission");
710   ink (white);
711   abort := true
712 END ;
713
714 PROCEDURE transmit_block;
715
716 VAR ch : CHAR ;
717     discard,
718     i : INTEGER ;
719
720 PROCEDURE get_ack;
721
722 BEGIN
723   ch := next_char (10);
724   IF timeout THEN
725     BEGIN
726     start_error;
727     WRITE ("Timeout on ACK");
728     error
729     END
730   ELSE
731   IF ch = can THEN
732     process_can
733   ELSE
734   IF ch <> ack THEN
735     BEGIN
736     start_error;
737     WRITE ("Got ",ch," for ACK");
738     error
739     END
740 END ;  (* of get_ack *)
741
742 BEGIN
743 sum_check := 0;
744 FOR i := 0 TO 127 DO
745   BEGIN
746   ch := MEMC [next_address];
747   next_address := next_address + 1;
748   sum_check := sum_check + ch;
749   buffer [i] := ch
750   END ;
751 IF display_file THEN
752 FOR i := 0 TO 127 DO
753   display_char (buffer [i])
754 ELSE
755   WRITE ("*");
756 IF want_crck THEN
757   BEGIN
758   sum_check_2 := calc_crck;
759   sum_check := sum_check_2 SHR 8;
760   sum_check_2 := sum_check_2 AND $ff
761   END ;
762 retries := 0;
763 inverse_block_no := block_no XOR $ff;
764 expected_block := block_no;
765 REPEAT
766   to_modem (soh);  (* start block *)
767   to_modem (block_no);
768   to_modem (inverse_block_no);
769   FOR i := 0 TO 127 DO
770     BEGIN
771     discard := from_modem;
772     to_modem (buffer[i])
773     END ;
774   to_modem (sum_check);
775   IF want_crck THEN
776     to_modem (sum_check_2);
777   get_ack
778 UNTIL abort
779 OR ((NOT timeout) AND (ch = ack));
780 IF next_address >= final_address THEN
781 IF NOT abort THEN
782   BEGIN
783   retries := 0;
784   expected_block := -1;
785   REPEAT
```

```
786     to_modem (eot);
787     get_ack
788   UNTIL abort OR ((NOT timeout) AND (ch = ack));
789   IF NOT abort THEN
790     eof := true
791   END ;
792 block_no := block_no + 1
793 END ;
794
795 PROCEDURE send_file;
796
797 VAR ch : CHAR ;
798 BEGIN
799 WRITELN ;
800 IF final_address = start_address THEN
801   BEGIN
802   start_error;
803   WRITELN ("No file loaded.");
804   command := " ";  (* menu again *)
805   END
806 ELSE
807   BEGIN
808   ink (light_green);
809   WRITELN ("[Send a File]");
810   ink (green);
811   analyse_file (false);
812   next_address := start_address;
813   block_no := 1;
814   expected_block := 1;
815   abort := false;
816   eof := false;
817   retries := 0;
818   purge;  (* empty buffer *)
819   WRITELN ; WRITELN ;
820   WRITELN ("Awaiting initial NAK");
821   REPEAT
822     ch := next_char (60);  (* wait a minute *)
823     IF timeout THEN
824       BEGIN
825       start_error;
826       WRITELN ("No response from other end")
827       END
828     ELSE
829       BEGIN
830       IF ch = nak THEN
831         want_crck := false
832       ELSE
833         IF ch = "c" THEN
834           want_crck := true
835         ELSE
836           IF ch = can THEN
837             process_can
838           ELSE
839             BEGIN
840             start_error;
841             WRITE ("Got ",ch," for NAK");
842             error
843             END
844       END
845   UNTIL (ch = nak) OR (ch = "c")
846      OR timeout OR abort;
847   IF NOT (timeout OR abort) THEN
848     REPEAT
849       transmit_block
850     UNTIL abort OR eof;
851   IF eof THEN
852     BEGIN
853     WRITELN ;
854     WRITELN ("File transmitted successfully")
855     END
856   ELSE
857     cancel_trans
858   END
859 END ;
860
861 PROCEDURE save_in_memory (x);
862
863 BEGIN
864 IF capture THEN
865   BEGIN
866   IF final_address = $7c00 THEN
867     BEGIN
868     WRITELN ;
869     WRITELN ;
870     start_error;
871     WRITELN
872     ("** Memory Buffer Almost Full **");
873     WRITELN ;
874     WRITELN ;
875     ink (white)
876     END ;
877   IF final_address < $8000 THEN
878     BEGIN
879     MEMC [final_address] := x;
880     final_address := final_address + 1
881     END
882   END
883 END ;
884
885 PROCEDURE terminal_mode (half_duplex);
886
887 CONST active = 7;
888 VAR input : CHAR ;
889     x : INTEGER ;
890 BEGIN
891 last_terminal_mode := command;
892 WRITELN ;
893 ink (light_green);
894 WRITE ("Terminal Mode - ");
895 IF half_duplex THEN
896   WRITE ("Half")
897 ELSE
898   WRITE ("Full");
899 WRITELN (" duplex");
900 WRITELN
901 ("Press <Commodore> key for Main Menu");
902 WRITELN ;
903 ink (white);
904 SPRITE (1, active, true);
905 REPEAT
906   x := CURSORX ;
907   IF x > 40 THEN
908     x := x - 40;
909   POSITIONSPRITE (1,
910     x * 8,
911     CURSORY * 8 + 42);
912   input := from_modem;
913   IF input <> 0 THEN
914     BEGIN
915     display_char (input);
916     save_in_memory (input)
917     END ;
918   input := GETKEY ;
919   IF input <> 0 THEN
920     BEGIN
921     IF (input >= $c1) AND
922        (input <= $da) THEN
923       input := input - $60;
924     IF input = $8d THEN
925        input := cr
926      ELSE
927        IF (input = $9d)
928        OR (input = $14) THEN
929          input := bs
930        ELSE
931          IF input = 29 THEN
932            input := fs
933          ELSE
934            IF input = home THEN
935              input := ff
936            ELSE
937              IF input = 133 THEN
938                BEGIN
939                CLOSE (2);
940                MEMC [$dd00] := 0;
941                SOUND (3, 10);
942                open_rs232_file;
943                input := cr
944                END ;
945
946 (* Reverse upper/lower case *)
947
948     IF (input >= $61) AND
949        (input <= $7a) THEN
950        input := input - $20
951     ELSE
952     IF (input >= "a") AND
953        (input <= "z") THEN
954        input := input + $20;
955     to_modem (input);
956     IF half_duplex THEN
957       BEGIN
958       save_in_memory (input);
959       ink (light_blue);
```

```
960       display_char (input);
961       ink (white)
962       END
963     END
964 UNTIL commodore_logo;
965 SPRITE (1, active, false)
966 END ;
967
968 PROCEDURE type_file (printit);
969
970 BEGIN
971 next_address := start_address;
972 WRITELN ;
973 WRITELN
974 ("Press <Commodore> key to abort list");
975 WRITELN
976 ("      <SHIFT>     key to pause list");
977 WRITELN ;
978 IF printit THEN
979   BEGIN
980   OPEN (4, printer_channel,
981            printer_sec_addr,
982            " ");
983 (* ** special printer stuff
984       goes here ***   *)
985   PUT (4)
986   END ;
987 ink (light_green);
988 WHILE (next_address < final_address)
989   AND NOT commodore_logo DO
990   BEGIN
991   REPEAT
992   UNTIL NOT shift_key_pressed;
993   display_char (MEMC [next_address]);
994   next_address := next_address + 1
995   END ;
996 IF printit THEN
997   BEGIN
998   PUT (0);
999   CLOSE (4)
1000  END ;
1001 WRITELN
1002 END ;
1003
1004 PROCEDURE erase_buffer;
1005
1006 VAR reply : CHAR ;
1007 BEGIN
1008 IF final_address <> start_address THEN
1009   BEGIN
1010   start_error;
1011   WRITE ("Erase conversation? (",
1012     final_address - start_address,
1013     " bytes) Y/N ");
1014   REPEAT
1015     READ (reply);
1016     reply := reply AND $7f
1017   UNTIL (reply = "y")
1018      OR (reply = "n");
1019   WRITELN (CHR (reply));
1020   IF reply = "y" THEN
1021     final_address := start_address
1022   ELSE
1023     command := " "
1024   END
1025 END ;
1026
1027 PROCEDURE memory_on_off;
1028
1029 BEGIN
1030 WRITELN ;
1031 ink (light_green);
1032 capture := NOT capture;
1033 WRITE ("Conversations now ");
1034 IF NOT capture THEN
1035   WRITE ("not ");
1036 WRITELN ("saved in memory.")
1037 END ;
1038
1039 BEGIN
1040   init;
1041   REPEAT
1042     ink (green);
1043     CASE command OF
1044      "a": analyse_file (false);
1045      "l": analyse_file (true);
1046      "c": cancel_trans;
1047      "d":
1048      IF final_address <> start_address
1049        THEN save_nominated_file;
1050      "e": erase_buffer;
1051      "f": terminal_mode (false);
1052      "m": memory_on_off;
1053      "h": terminal_mode (true);
1054      "r": receive_file;
1055      "s": send_file;
1056      "p": type_file (true);
1057      "t": type_file (false)
1058     END ;  (* of case *)
1059     IF (command = "s")
1060     OR (command = "r") THEN
1061       command := last_terminal_mode
1062     ELSE
1063       BEGIN
1064       ink (green);
1065       WRITELN (CHR (14));  (* lower case *)
1066       WRITELN
1067       ("<A>nalyse memory");
1068       WRITELN
1069       ("<C>ancel transmission");
1070       WRITELN
1071       ("<D>ump conversation to disk/cassette");
1072       WRITELN
1073       ("<E>rase last conversation");
1074       WRITELN
1075       ("<F>ull duplex terminal");
1076       WRITELN
1077       ("<H>alf duplex terminal");
1078       WRITELN
1079       ("<L>oad file");
1080       WRITELN
1081       ("<M>emory capture off/on");
1082       WRITELN
1083       ("<P>rint last file");
1084       WRITELN
1085       ("<R>eceive a file");
1086       WRITELN
1087       ("<S>end a file");
1088       WRITELN
1089       ("<T>ype last file");
1090       WRITELN
1091       ("<Q>uit program");
1092       WRITELN ;
1093       WRITE ("Command? < >",CHR (157),CHR (157));
1094       ink (light_blue);
1095       REPEAT
1096         READ (command);
1097         command := command AND $7f
1098       UNTIL (command = "f")
1099          OR (command = "s")
1100          OR (command = "q")
1101          OR (command = "l")
1102          OR (command = "m")
1103          OR (command = "e")
1104          OR (command = "c")
1105          OR (command = "h")
1106          OR (command = "t")
1107          OR (command = "a")
1108          OR (command = "p")
1109          OR (command = "d")
1110          OR (command = "r");
1111       WRITELN (CHR (command))
1112       END ;
1113     CASE command OF
1114      "q", "l", "r" : erase_buffer
1115     END  (* of case *)
1116   UNTIL (command = "q");
1117   CLOSE (2)
1118 END .
```

PRETTY-PRINT PROGRAM

"Pretty-printing" is the (rather strange) name given to programs which read in a source program and format them in a standard way. In our case the G-Pascal pretty-print program will read in any reasonable size G-Pascal program and output a properly formatted version (BEGINs and ENDs lined up under each other, IFs indented and so on).

The program loads the original source file at address $1200, and puts the converted file at address $4000, so that the converted version can be examined in memory using the editor, and then saved to disk if desired, in the normal way.

Because of this, the largest program that can be converted is 11.5K - larger programs will behave strangely during the conversion process.

The converted program is displayed on the screen at the same time as it is being written to memory, so you can watch the (rather slow) process as it goes, and halt it (by pressing RUN/STOP) if something looks wrong.

The program assumes the file name of the program to be converted is "filename" - to change this alter line 88 before compiling.

You can of course change the conversion parameters by examining and altering the program to your own requirements. The main design criteria of the converter was to allow it to quickly and easily format large, badly laid-out programs. You may wish to add a bit of 'customised' formatting to the final version before saving it to disk.

The pretty-print program is also interesting as it illustrates the internal format of G-Pascal programs, and gives the equivalent token values of most of the G-Pascal reserved words.

```
 1 (* G-Pascal Pretty-Print program
 2
 3    Author: Nick Gammon
 4
 5 *)
 6
 7 (* %a $810  *)
 8
 9 CONST dle = $10;
10       start_address = $1200;
11       cr = 13;
12       true = 1;
13       false = 0;
14       xreg = $2b3;
15       yreg = $2b4;
16 VAR old,
17     new,
18     indent,
19     temp_indent,
20     col,
21     got_dle,
22     final_address,
23     line,
24     newline : INTEGER ;
25     ch : CHAR ;
26
27 PROCEDURE output (x);
28 (******************)
29 VAR i : INTEGER ;
30 BEGIN
31   MEMC [new] := x;
32   new := new + 1;
33   IF got_dle THEN
34     BEGIN
35     got_dle := false;
36     FOR i := 1 TO (x AND $7f) DO
37       WRITE (" ")
38     END
39   ELSE
40     IF x = dle THEN
41       got_dle := true
42     ELSE
43       WRITE (CHR (x));
44   IF x = cr THEN
45     BEGIN
46     WRITE (line,": ");
47     line := line + 1
48     END
49 END ;
50
51 PROCEDURE get_char;
52 (******************)
53 BEGIN
54   ch := MEMC [old];
55   IF ch = """" THEN
56     REPEAT
57       MEMC [$49] := 12;
58       output (ch);
59       col := col + 1;
60       old := old + 1;
61       ch := MEMC [old]
62     UNTIL (ch = """") OR (ch = cr);
63   IF (ch = "(")
64    AND (MEMC [old + 1] = "*") THEN
65     REPEAT
66       newline := true;
67       MEMC [$49] := 12;
68       output (ch);
69       col := col + 1;
70       old := old + 1;
71       ch := MEMC [old]
72     UNTIL (MEMC [old - 1] = ")")
73       AND (MEMC [old - 2] = "*");
74   WHILE ch = cr DO
75     BEGIN
76     old := old + 1;
77     ch := MEMC [old]
78     END ;
79   WHILE ch = dle DO
80     BEGIN
81     old := old + 1;
82     ch := " "
83     END ;
84   MEMC [$49] := 0
85 END ;
86
87 BEGIN
88 LOAD (8, start_address, 0, "filename");
89 final_address := MEMC [xreg]
90                 + MEMC [yreg] SHL 8;
91 WRITELN ;
92 WRITELN ("End address was: ",
93     HEX (final_address));
94 WRITELN ;
95 old := start_address;
96 new := $4000;
97 indent := 0;
98 col := 0;
99 WRITE ("1: ");
100 line := 2;
101 got_dle := false;
102 temp_indent := false;
103 newline := false;
104 WHILE MEMC [old] <> 0 DO
105   BEGIN
106   get_char;
107   IF ch <> 0 THEN
108     BEGIN
109     CASE ch OF
110       $84,  (* array *)
```

```
      $85,  (* of *)
      $8a,  (* or *)
      $8b,  (* div *)
      $8c,  (* mod *)
      $8d,  (* and *)
      $8e,  (* shl *)
      $8f,  (* shr *)
      $90,  (* not *)
      $91,  (* mem *)
      $93,  (* then *)
      $97,  (* do *)
      $9b,  (* to *)
      $fe,  (* integer *)
      $a1,  (* char *)
      $a2,  (* memc *)
      $a4,  (* xor *)
      $a7,  (* getkey *)
      $a9,  (* address *)
      $e6,  (* spritecollide *)
      $e7,  (* groundcollide *)
      $e8,  (* cursorx *)
      $e9,  (* cursory *)
      $ea,  (* clock *)
      $eb,  (* paddle *)
      $ec,  (* spritex *)
      $ee,  (* spritey *)
      $ef,  (* random *)
      $f0,  (* envelope *)
      $f1,  (* scrollx *)
      $f2,  (* scrolly *)
      $f3,  (* spritestatus *)
      $f8,  (* abs *)
      $f9,  (* invalid *)
      $fd : (* freezestatus *)
      BEGIN END
    ELSE
    IF ((ch < $b0) OR (ch > $de))
    AND (ch > $81) THEN
      newline := true
    END ;  (* of case *)
    IF ch = $88  (* begin *)
      THEN
        temp_indent := false;
    CASE ch OF
      $89,  (* end *)
      $99   (* until *)
        : indent := indent - 2;
      $86,  (* procedure *)
      $87,  (* function *)
      $82,  (* const *)
      $83   (* var *)
        : indent := 0
    END  (* of case *) ;
    CASE ch OF
      $82,  (* const *)
      $83,  (* var *)
      $86,  (* procedure *)
      $87   (* function *)
       : output (cr)
    END ;  (* of case *)
    IF newline THEN
      BEGIN
      output (cr);
      col := 1;
      WHILE ch = " " DO
        BEGIN
        old := old + 1;
        get_char
        END ;
      IF temp_indent THEN
        indent := indent + 2;
      IF indent > 0 THEN
        BEGIN
        output (dle);
        output (indent OR $80)
        END ;
      col := col + indent;
      IF temp_indent THEN
        indent := indent - 2
      END ;
    newline := false;
    output (ch);
```

```
    CASE ch OF
      $85, $8a, $92, $97, $9b:
         col := col + 2;
      $83, $89, $8b, $8c, $8d, $8e,
      $8f, $90, $91, $9a, $a4, $f8:
         col := col + 3;
      $93, $94, $95, $9e, $9f, $a1,
      $a2, $a6, $aa, $fa, $fb:
         col := col + 4;
      $82, $84, $88, $96, $99, $9d,
      $a8, $e1, $e3, $ea:
         col := col + 5;
      $98, $9c, $a3, $a7, $df, $e5,
      $eb, $ef:
         col := col + 6;
      $fe, $a9, $e8, $e9, $ec, $ee,
      $f1, $f2, $f9:
         col := col + 7;
      $87, $e2, $e4, $ed, $f0:
         col :=col + 8;
      $86: col := col + 9;
      $f4, $f5: col := col + 10;
      $f6: col := col + 11;
      $a5, $f3, $fc, $fd:
         col := col + 12;
      $e6, $e7, $f7: col := col + 13;
      $e0: col := col + 14
    ELSE
      col := col + 1
    END ;  (* of case *)
    CASE ch OF
      $88,  (* begin *)
      $95,  (* case *)
      $98   (* repeat *)
        : indent := indent + 2
    END  (* of case *) ;
    IF indent < 0 THEN
      indent := 0;
    CASE ch OF
     ";",
     $88,  (* begin *)
     $89,  (* end *)
     $93,  (* then *)
     $94,  (* else *)
     $85,  (* of *)
     $98   (* repeat *)
      : newline := true
    END ;  (* of case *)
    IF (ch = $89)  (* end *)
    AND (MEMC [old + 1] = ";") THEN
      newline := false;
    IF (ch = $85)  (* of *)
    AND ((MEMC [old + 1] = $fe) (* integer *)
     OR  (MEMC [old + 1] = $a1)) (* char *)
     THEN newline := false;
    temp_indent := false;
    IF (ch = $93)  (* then *)
    OR (ch = $97)  (* do *)
    OR (ch = $94)  (* else *)
      THEN temp_indent := true;
    old := old + 1;
    IF (NOT newline)
    AND (col > 70) THEN
      CASE ch OF
      " ", ")", "*", "+", ",",
      "-", "/", ":", "<", "=",
      ">", "]":
        BEGIN
        newline := true;
        CASE MEMC [old] OF
          "=", ">", ")", ";":
            newline := false
        ELSE
          temp_indent := true
        END  (* of case *)
        END
      END  (* of case *)
    END
  END ;
  output (cr);
  output (0);
  WRITELN ;
  WRITELN ("Final address = ",HEX (new))
END .
```

ADDRESS LIST PROGRAM

In response to a number of requests from G-Pascal owners we have produced a random disk file accessing program. This program illustrates:

- Modular coding techniques.
- How to open and read a random file.
- How to read and write strings.
- How to read the disk error channel.
- Displaying a 'default' reply and accepting a response.
- Random disk accessing.
- Using the CURSOR statement for full-screen displays.

When you have keyed in the program, compiled and run it, it will open the file and display:

```
<R>ead, <C>hange, <N>ext or <Q>uit?
```

As currently set up the program will allow you to read and change any one of 100 records, numbered from 1 to 100. The current record number is displayed near the top of the screen. Initially all records are set to blanks (in effect). To change a record, first read it in. When you select 'R' (for Read) the program will ask:

```
Record number?
```

Enter a record number from 1 to 100 and press RETURN. The current contents of that record (if any) will be displayed. To alter it select 'C' (for Change). The cursor will be positioned successively over the 4 lines of the record. Each line is 30 characters long. Under the line being changed will be displayed 30 hyphens to help you visualise how much space you have. To accept the current entry just press RETURN. Otherwise key in a new line and press RETURN, or make alterations to the existing one (using the cursor control keys) and press RETURN.

Once you have changed all four lines the new record will be written back to disk. You can now read and possibly change a new record.

Entering 'N' (for Next) will just step sequentially through the file.

You must exit from the program in an orderly fashion by pressing 'Q' (for Quit), otherwise the last disk buffer will not be written back to the disk file, and some of your changes will be lost.

You can change the size of each line by altering 'entry_length' (line 12) but you must also make a corresponding change to 'record_length' (line 10). Record_length is equal to ((entry_length + 1) * 4) + 1.

You can change the number of lines in a record by making straightforward changes to the appropriate parts of code. The contents of each line is free-format - that is, up to you. We suggest, for example, that line 1 would be a name, lines 2 and 3 the address, and line 4 a phone number, birthdate etc.

You will probably be able to think of simple improvements to the program - for example an ability to search each record for a particular name, or to print a record on your printer.

By changing 'max_record' (line 17) you can alter the maximum size of your file. Be careful if you make the value large, as the disk drive takes quite a while to initialise your file when you first access records at the end of the file. The main reason for having 'max_record' is to catch keying errors when entering the record number.

The file name on disk will be '0:mailing list' - you can change this by changing lines 78 to 87. The reason the file was opened this way was because of the need to include the record length as a binary number, rather than as ASCII characters.

```
 1 (* G-Pascal Mailing List
 2      using relative files.
 3    Author:  Nick Gammon
 4    (Gambit Games) - January 1985 *)
 5
 6 CONST true = 1;
 7       false = 0;
 8       cr = 13;
 9       clearscreen = 147;
10       record_length = 125;
11       name_length = 17;
12       entry_length = 30;
13
14       (* if entry_length changes,
15          also adjust record_length *)
16
17       max_record = 100;
18
19 VAR file : ARRAY [name_length] OF CHAR ;
20     result : ARRAY [80] OF CHAR ;
21     name1, name2, name3, name4
22       : ARRAY [entry_length] OF CHAR ;
23     reply : CHAR ;
24     error,
25     result_code,
26     record_number : INTEGER ;
27
28
29 PROCEDURE read_error_channel;
30 (***************************)
31 BEGIN
32   GET (15);
33   READ (result);
34   GET (0);
35   result_code := (result [0] - "0") * 10
36            +  (result [1] - "0")
37 END ;
38
39 PROCEDURE display_array (addr);
40 (*****************************)
41 BEGIN
42   REPEAT
43     WRITE (CHR (MEMC [addr]));
44     addr := addr - 1    (* next *)
45   UNTIL MEMC [addr + 1] = cr
46 END ;
47
48 PROCEDURE change (col, old_data);
49 (*******************************)
50 VAR i : INTEGER ;
51 BEGIN
52 CURSOR (col + 1, 1);
53 FOR i := 1 TO entry_length DO
```

```
  WRITE ("-");
CURSOR (col, 1);
display_array (old_data);
CURSOR (col, 1)
END ;

PROCEDURE open_relative_file;
(**************************)
CONST
    areg = $2b2;
    xreg = $2b3;
    yreg = $2b4;
    cc   = $2b1;
    openit = $ffc0;
    setlfs = $ffba;
    setnam = $ffbd;
    readst = $ffb7;
BEGIN
  (* open error channel *)

  OPEN (15, 8, 15, "i");

  (* set up file name *)

  file [17] := "0"; file [8] := " ";
  file [16] := ":"; file [7] := "l";
  file [15] := "m"; file [6] := "i";
  file [14] := "a"; file [5] := "s";
  file [13] := "i"; file [4] := "t";
  file [12] := "l"; file [3] := ",";
  file [11] := "i"; file [2] := "l";
  file [10] := "n"; file [1] := ",";
  file [ 9] := "g"; file [0] :=
                       record_length;

  (* open file *)

  MEMC [areg] := 2; (* file *)
  MEMC [xreg] := 8; (* disk *)
  MEMC [yreg] := 3; (* channel *)
  CALL (setlfs);
  MEMC [areg] := name_length + 1;
  MEMC [xreg] := ADDRESS
     (file [name_length]);
  MEMC [yreg] := ADDRESS
     (file [name_length]) SHR 8;
  CALL (setnam);
  CALL (openit);

  (* check result of open *)

  IF MEMC [cc] AND 1 THEN
    error := MEMC [areg]
  ELSE
    BEGIN
    CALL (readst);
    error := MEMC [areg] AND $bf
    END ;
  read_error_channel;
  IF (error <> 0)
  OR (result_code <> 0) THEN
    BEGIN
    WRITELN ("Error code: ",error);
    display_array (ADDRESS (result [0]));
    WRITELN ("Error on Open.");
    CLOSE (2);
    CLOSE (15);
    error := error / 0  (* force abort *)
    END
END ;

PROCEDURE position_file;
(**********************)
BEGIN
  CURSOR (25, 1);
  WRITE ("Please wait ...");
  CURSOR (1, 1);
  PUT (15);
  WRITE ("p",CHR (3), (* channel *)
         CHR (record_number AND $ff),
         CHR (record_number SHR 8),
         CHR (0));  (* position in record *)
  PUT (0);
  read_error_channel
END ;

PROCEDURE read_record;
(**********************)
BEGIN
  IF (result_code = 0) THEN
    BEGIN
    GET (2);
    READ (name1);
    GET (0)
    END ;
  IF (result_code <> 0) OR
    (name1 [0] = 255) THEN
    BEGIN
    name1 [0] := cr;
    name2 [0] := cr;
    name3 [0] := cr;
    name4 [0] := cr
    END
  ELSE
    BEGIN
    GET (2);
    READ (name2, name3, name4);
    GET (0)
    END
END ;

PROCEDURE display_record;
(**********************)
BEGIN
  WRITELN (CHR (clearscreen),
  "----Name and Address Filing System----");
  CURSOR (4,1);
  WRITELN ("Record number: ",
           record_number);
  CURSOR (8, 1);
  display_array (ADDRESS (name1 [0]));
  CURSOR (10, 1);
  display_array (ADDRESS (name2 [0]));
  CURSOR (12, 1);
  display_array (ADDRESS (name3 [0]));
  CURSOR (14, 1);
  display_array (ADDRESS (name4 [0]))
END ;

PROCEDURE get_record_key;
(**********************)
BEGIN
REPEAT
  CURSOR (22, 1);
  WRITE ("Record number? ");
  READ (record_number);
  IF (record_number < 1)
  OR (record_number > max_record) THEN
    WRITELN
    ("Record number must be from 1 to ",
       max_record)
UNTIL (record_number >= 1)
  AND (record_number <= max_record)
END ;

PROCEDURE change_record;
(**********************)
BEGIN
  change (8, ADDRESS (name1 [0]));
  READ (name1);
  name1 [entry_length] := cr;
  change (10, ADDRESS (name2 [0]));
  READ (name2);
  name2 [entry_length] := cr;
  change (12, ADDRESS (name3 [0]));
  READ (name3);
  name3 [entry_length] := cr;
  change (14, ADDRESS (name4 [0]));
  READ (name4);
  name4 [entry_length] := cr;
  position_file;
  PUT (2);
  display_array (ADDRESS (name1 [0]));
  display_array (ADDRESS (name2 [0]));
  display_array (ADDRESS (name3 [0]));
  display_array (ADDRESS (name4 [0]));
  PUT (0)
END ;

(*
****** MAIN PROGRAM STARTS HERE ******
*)
BEGIN  (* main program *)
WRITE (CHR (clearscreen),
    "Opening mailing list file.");
open_relative_file;
```

```
230 record_number := 1;
231 REPEAT
232   position_file;
233   read_record;
234   display_record;
235   CURSOR (20, 1);
236   WRITE
237   ("<R>ead, <C>hange, <N>ext or <Q>uit? ");
238   REPEAT
239     READ (reply);
240      (* convert to lower case *)
241     reply := reply AND $7f
242   UNTIL (reply = "r")
243      OR (reply = "n")
244      OR (reply = "c")
245      OR (reply = "q");
246   WRITELN (CHR (reply));
247   WRITELN ;
248   CASE reply OF
249   "r": get_record_key;
250   "n":      (* read next record *)
251     IF record_number < max_record THEN
252       record_number := record_number + 1;
253   "c": change_record
254   END ;  (* of case *)
255 UNTIL reply = "q";
256 CLOSE (2);
257 CLOSE (15)
258 END .
```

QUADRANTIAN FIGHTER

As mentioned in an earlier letter to customers, Gambit Games is marketing an arcade game - Quadrantian Fighter - written entirely in G-Pascal without using MEM, MEMC, or CALL. Quadrantian Fighter makes extensive use of sprites including MOVESPRITE and ANIMATESPRITE and the other sprite-handling statements. It also incorporates sound effects, and smooth-scrolling instructions.

It is supplied in two forms - a compressed one for fast loading and playing, and an uncompressed form (one statement per line) for easier examination and changing. Both function identically. The uncompressed form consists of 1,283 source statements.

The game is a good illustration of how to program an arcade game entirely in a high-level language. Its presentation and execution speed are comparable to machine-code games.

Quadrantian Fighter is available from Gambit Games for $9.95 plus $2 postage and packaging. Full source code is supplied.

Quadrantian Fighter is written by Wayne Morellini - a G-Pascal owner from Queensland.

STORING LARGER PROGRAMS

It is possible to store a larger G-Pascal source program than the maximum of 16K mentioned on page 77 of the G-Pascal manual. To do this, you alter the lower limit of the G-Pascal source code from $4000 (which is its normal position) to anywhere below that, down to $800 which is just past where screen memory ends. If you change the start address of the program to $800 then your program can be up to 30K in length.

There are some restrictions, however. The area of memory between $800 and $4000 was set aside for sprite definitions, bit-mapped graphics, machine code, P-codes (if required), extra screen pages (for page flipping) and user-defined character sets. If you make the source start at $800 then **none** of those other uses are available to you - this may be alright if you are just writing, say, a text-only adventure game, or some text-only educational software. Otherwise you will need to compromise. For example, if you make the source start at address $2000 then you still have from address $800 to $2000 for sprites etc., and your program can now be 24K in length.

To change the source start address enter, compile, and run the following program (or similar):

```
1 BEGIN MEMC [$800a] := $08 END .
```

The '$08' could be any value from 08 to 40 (the low order byte is assumed to be zero, so '$08' represents address $0800 in this case). To change back to normal operation, run the program with the value being $40.

After running this program you may see rubbish on the screen if you do an editor List. To get rid of that, just enter:

```
d 1,9999
```

in the Editor - this will give you an empty source file.

If you have saved to disk or cassette a program larger than 16K in length, then make sure that you patch the source start address (as above) **before** reloading the program in a subsequent session of G-Pascal. **Loading a program larger than 16K to an unpatched version of G-Pascal will clobber the compiler.**

HOW TO QUICKLY LOAD G-PASCAL

The smallest number of keystrokes needed to load G-Pascal from disk (after turning on your C64's power) is:

```
L <SHIFT>0 "*",8
```

Typing L followed by SHIFT/O is a standard Basic abbreviation for LOAD. Using an asterisk for the filename tells the disk drive to load the first file found on the disk ("*" is actually a 'wild-card' specification). As the compiler is the first program on the disk (at least on the ones Gambit Games supply) then this loads G-Pascal with only 8 keystrokes (including RETURN). Once it has loaded you can type RUN, or to save one keystroke, R SHIFT/U which is the abbreviation for RUN (followed by RETURN of course).

PROGRAMS AVAILABLE ON DISK

As a service to readers Gambit Games will make available a disk containing the three major programs in this issue (MODEM64 version 1.6, Pretty-print, and Mailing List) for $10 including postage. These programs between them consist of 1,652 source statements which will take a while to key in from scratch. If you are interested send a cheque, postal order or credit card number to Gambit Games requesting the disk for the January 1985 issue of G-Pascal News.

NEXT ISSUE

In our next issue (April) we plan to include further program listings, syntax diagrams, and start our G-Pascal tutorial. We would appreciate hearing from readers who would like to see particular subjects discussed, or who can suggest the 'level' at which tutorial material should start. There is still time to influence the next issue (until the middle of March, 1985) so write now.

COPYRIGHT

DEFAULT DISK OR CASSETTE ACCESS

In normal operation G-Pascal always asks you whether to load or save a file from/to disk or cassette, and then asks the file name. If you are always planning to use either disk or cassette (but not both), and find always answering the question 'Disk or Cassette' irritating, you can make a small 'patch' that will bypass the question and set up the correct default. The program to achieve this is:

Disk

```
BEGIN MEM [$9c66] := $ea44a9 END .
```

Cassette

```
BEGIN MEM [$9c66] := $ea43a9 END .
```

Note the use of the word MEM rather than MEMC as three bytes are being patched.

The actual code being generated here is:

```
LDA #'D'    (or 'C')
NOP
```

which replaces a subroutine call to 'get a reply' and return with the value entered.

SUPERSPRITE EDITOR

SuperSprite Editor is a useful and powerful sprite editing tool. It can hold many sprite shapes in memory simultaneously for comparison, copying and editing. You can easily animate (sequence) through series of sprite shapes to test animation effects. There are 22 sample sprites supplied with the editor.

SuperSprite Editor is available from Gambit Games for $11.95 plus $2 postage and packaging. Full source code is supplied.

SuperSprite Editor is written by Craig Brookes - a G-Pascal owner from Western Australia.

CREDITS

Material in this issue by Nick Gammon. Typeset on a Brother EM-100 typewriter. Typesetting software specially written in G-Pascal to handle 'proportional space' typefaces. Editorial assistance from Sue Gobbett and Cynthia Gammon. Written and printed in January 1985.