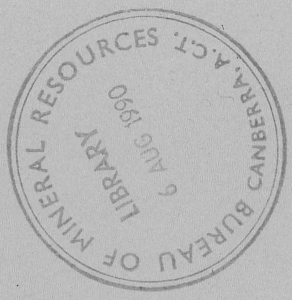


Bureau of Mineral Resources, Geology & Geophysics

R E C O R D

BMR RECORD 1990/39

GetSeq

A PC Program for Extracting Data from a Remote SQL Database - An Example of Client-Server Programming

by

R.J. Ryburn

BMR RECORD 1990/39

GetSeq

A PC Program for Extracting Data from a Remote SQL Database - An Example of Client-Server Programming

by

R.J. Ryburn

11/07/1990

CONTENTS

1 - INTRODUCTION

Getseq is a product of BMR Project 520.02, 'Integration of Geoscientific Data Sets'.

Data storage and retrieval methods are currently undergoing a revolution that has implications for geoscientists. Most interactive applications are moving off centralized computers to the friendlier environments on PCs and workstations, leaving the data-heavy tasks such as database management to larger computers and data servers. The problem now arises: how do you write the specialized programs geoscientists need on their workstations, while retaining the data and data-manipulation functions on a remote 'mainframe' or file server?

GetSeq is an attempt to write a simple but useful IBM PC program that gets its data from a remote host. In the current BMR computing environment GetSeq's main function is to select data from an ORACLE database on the Data General MV 20 000 computer and to download it to a PC file over the Sytec LAN. It can also be used to rapidly download DG files to a PC. The required Data General and ORACLE logon/logoff is handled quickly and automatically. However, GetSeq is not restricted to this environment. It can be used on any IBM clone needing data from a remote host over an asynchronous serial line or modem. GetSeq may eventually be extended to operate over Ethernet LANs.

A major purpose of GetSeq, though, is to demonstrate the essential programming techniques required for a PC program to communicate with a remote host, and to this end a listing of GetSeq is included in Appendix B. Getseq is written in Microsoft C version 5.1, and incorporates a few prepackaged functions from Greenleaf Software's communication library.

GetSeq will be followed by PlotSeq, a program for PC plotting of graphs from data held on a remote host, and SQL databases in particular. With this program as a starting point, programs for specialised scientific graphs, such as concordia diagrams for geochronology, can be developed quickly and cheaply.

Those wishing to use GetSeq should obtain copies on floppy disk of the program (GETSEQ.EXE) and test logon file (GSLOG.ON) from Rod Ryburn, Room 79, Ground Floor, BMR.

2 - A QUICK DEMONSTRATION

You must have an IBM-compatible PC connected to BMR's Sytek LAN to run GetSeq, and to start with, the PC must be logged right off the LAN. Insert the GetSeq floppy disk in drive A and select it by typing -

A: <ENTER>.

To run GetSeq in query mode type -

GS ? <ENTER>

GetSeq now proceeds to log on to the LAN, the Data General computer and Oracle. Should the automatic logon fail, after a ten-second timeout you will be diverted to GetSeq's main menu in interactive mode (see section 4). If logon to both the DG and Oracle is successful you are asked to name a DOS capture file on your PC -

```
*  Liz Anderson (x9213) or David Downie (x9432)  *
*  if you need more information.                 *
*                                                *
**************************************************

Most recent logon       14-May-90       14:29:48

AOS/VS CLI   Rev 07.65.00.00    14-MAY-90       16:07:00
ROD.DBASE> SQLPLUS GEOCHEM

SQL*Plus: Version 2.0.14.5 - Production on Mon May 14 16:07:03 1990

Copyright (c) 1987, Oracle Corporation, California, USA.  All rights reserved.

Enter password:

I mean really enter your password! :

Connected to: ORACLE V5.1.22.5 - Production


SQL>

 Enter DOS pathname for capture file, or ESC to exit
 : TEMP
```

Enter ***TEMP*** to create a file on the current directory. If the file already exists you will be asked if you wish to overwrite the file (and thereby destroy it). If not, you can enter another file name.

You will now find yourself talking to Oracle's SQL*Plus, but under the control of GetSeq. Enter the following SQL select statement -

SELECT HMAPNAME FROM HMAPS WHERE N_LAT > 40.5; <ENTER>

The names of all 1:100 000 maps in Tasmania will now be selected from the publicly accessible HMAPS table, listed on the screen, and simultaneously stored on the file TEMP in your current DOS directory. Because the file TEMP is on a floppy disk the transfer will be slower than it would be to a file on the hard disk, but GetSeq manages the transfer in an orderly manner with the help of built-in X-ON/X-OFF protocol between the PC and the LAN box.

As soon as the listing has finished press ESCAPE (or ENTER) to log off automatically and exit the program to DOS. If you now type -

TYPE TEMP <ENTER>

- you will see the transferred data -

```
SQL> select hmapname from hmaps where N_LAT > 40.5;

HMAPNAME
--------------------
SANDY CAPE
ARTHUR RIVER
HELLYER
FORTH
TAMAR
...

...
STORM BAY
SOUTH WEST CAPE
SOUTH EAST CAPE

37 records selected.

SQL>
```

3 - THE LOGON FILE

The logon file, GSLOG.ON, contains all the information necessary for GetSeq to log on to the LAN, the remote host, and the SQL database system. The supplied file looks like this -

```
LOGON  t10 s13 r"#" s"dg"13 r"CALL COMPLETED TO" r")" w1
       s13 r"Username: " s"ROD.DBASE"13 r"Password: " p1
       r"ROD.DBASE>"
ORACLE s17 s"SQLPLUS GEOCHEM"13 r"password: " p2 r"SQL>"
LOGOFF s17,13,"bye"13"bye"13 r"logging on ****" s7,127
       r"#" s"do"13,13 r"CLOSED" r"#"
```

Use an editor such as WordStar (in non-document mode) to modify this file to you own DG and Oracle logons. The meaning of the lower case prefixes in this file are as follows -

t - set the logon timeout to the nunber of seconds appended

s - send the following string and/or character(s)

r - receive the following string and/or character(s) before proceeding. The timeout applies

p - pause to enter the first or second password

w - wait - do nothing for the number of seconds indicated

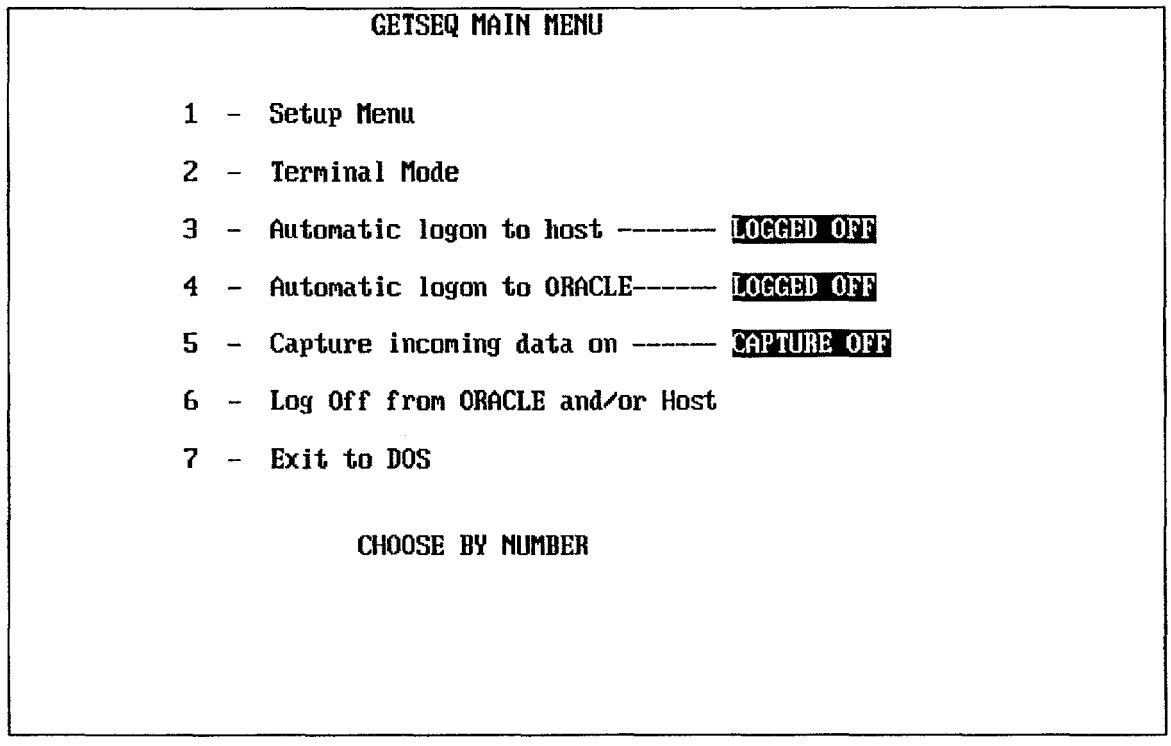
Note that the characters to be sent to the remote host can either be stated as a string of printable characters or the characters' ASCII decimal values. If the latter, then the numbers must be separated by commas or some other form of punctuation. In the supplied logon file the 13 specifies RETURN and 17 CONTROL-Q - in case output from the DG has been suspended.

4 - INTERACTIVE MODE

To run GetSeq in interactive mode type -

GS <ENTER>

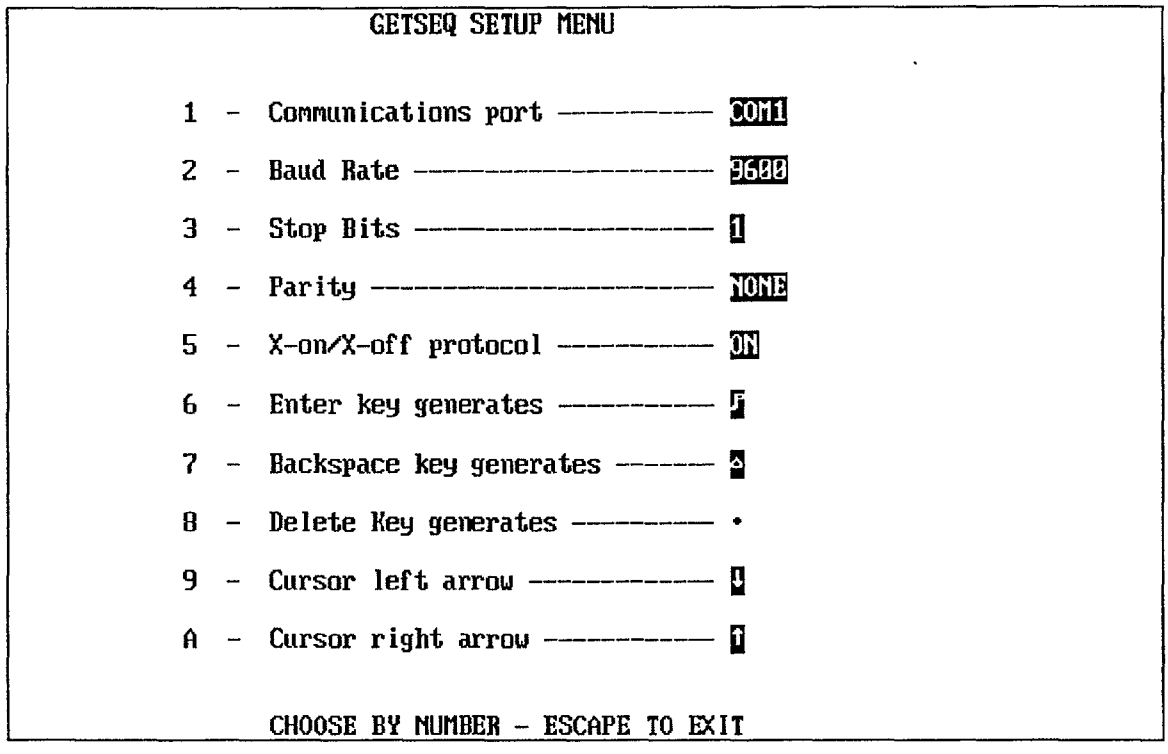
GetSeq's main menu appears -



The functions available from this menu are activated by the press of a number key. For example, you can press *5* to start data capture at any time, even before logging on. To stop data capture just press *5* again. To terminate GetSeq and return to DOS you may press *7* or enter ***ESCAPE***.

Before entering Terminal Mode you should use the automatic logon provided by items 3 or 4 in the menu, but your PC must first be off the LAN. If you engage Terminal Mode and log on manually GetSeq will not be aware of this and will not log you off automatically when you exit the program. Note that pressing *4* will first log you on to the LAN and the DG, if not already there. Should automatic logon fail you should enter Terminal Mode to make sure you are off the computer and LAN before trying again.

Press *1* to view the communication settings menu -



This menu provides you with control of serial communications and keyboard settings. The default values apply to the DG computer and should not need to be changed. Note that the characters or strings displayed against the adjustable keys are from the IBM-PC character set displayed in Appendix A.

5 - COMMAND MODE

There are various ways you can run GetSeq as a single DOS command, albeit with up to four command-line parameters. In all cases you must start with your PC logged right off the LAN (i.e., no LAN port open, athough you can edit the logon file to expect an open LAN port). Provided GetSeq completes the transfer, it always leaves the PC logged off the LAN when finished. If unsuccessful at any point it will exit to GetSeq's Main Menu. It is then up to the user to complete the transfer using Terminal Mode, and to log off in the correct manner.

5.1 DOWNLOADING A HOST FILE

For example, to retrieve an AOS (DG Operating System) file from the DG computer enter -

GS GEOCHEM:SQL:GETMAPS.SQL C:\DATA\MAPS2.ASC ZYX.WVU <ENTER>

The first parameter after *GS* is the AOS pathname of the DG file you wish to retrieve. The second parameter is the DOS pathname of the file on which the data is to be stored on your PC. *ZYX.WVU* is your optional DG password. GetSeq will prompt you for a password if it is not supplied as a parameter.

5.2 ALL DATA FROM AN ORACLE TABLE

To retrieve all data from an Oracle table you could type -

GS (GEOCHEM.REGIONS) A:REGIONS ZYXWVU CBA <ENTER>

In this case you are selecting all data from the table **GEOCHEM.REGIONS** and transferring it to a PC file called **REGIONS** on floppy drive A. **CBA** is your Oracle password. The round brackets enclosing the first parameter tell GetSeq that the parameter is an Oracle table name. The file produced by this method is unpaginated and has one heading at the beginning.

5.3 A SQL COMMAND AS A PARAMETER

You can also pass a complete SQL command as the second parameter. For example -

GS SELECT@@FROM@HMAPS@WHERE@MMAPID='SI55' CMAPS <ENTER>*

- will get you all data from the 1:100 000 maps table for the 1:1 000 000 map covering the Canberra 1:250 000 sheet area. There are some restrictions, however. '@' symbols must be used instead of spaces to ensure the whole command is passed as a single parameter. Because of DOS limitations, some non-alphanumeric characters cannot be used - for example, '>' and '<'. In the above case you are prompted for the passwords that were not included as command-line parameters.

5.4 SQL COMMANDS FROM A DOS FILE

Perhaps the most useful way of downloading Oracle data is to issue SQL*Plus commands from a DOS file. For example -

GS @SMALLCHM.SQL C:\GDA\CHEM.DAT ZYX.WVU CBA <ENTER>

The '@' symbol signals the pathname of a DOS file containing the SQL*Plus commands to be executed by Oracle on the DG computer. This parallels the way AOS files of SQL commands may be invoked from within SQL*Plus. In the above example, the DOS file SMALLCHM.SQL might contain the following SQL*Plus commands -

```
SET ECHO ON;
SET PAGESIZE 50000;
SET LINESIZE 80;
SET ARRAYSIZE 10;


SPOOL SMALLCHEM.LIS;

SELECT      SAMPLES.SAMPNO,
            STRATGROUP, SUBGROUP, STRATUNIT, ROCKNO, GROUPING, MAPSYMBOL,
            LITHOLOGY, MAPNO, GRIDREF, DRILLHOLE, DEPTH, AGE, OTHERDATA,
            SIO2, TIO2, AL203, FE203, FEO, MNO, MGO, CAO, NA2O, K2O, P2O5,
            H2OPLUS, H2OMIN, CO2, LOI,
            BA, LI, RB, SR, PB, TH, U, ZR, NB, Y, LA,
            CE, PR, ND, SC, V, CR, MN,
            CO, NI, CU, ZN, SN, W, MO, GA, ARS, S, C,
            F, CL, B, AG, AU, BI, GE, BE, SE, CS,
            DLAT, DLONG
    FROM SAMPLES, MAJORS, TRACES
    WHERE  samples.ORIGNO         = 29
    AND    REGIONO                = 16
    AND   ((samples.SAMPNO        = majors.SAMPNO (+))
           and (samples.ORIGNO    = majors.ORIGNO (+)))
    AND   ((samples.SAMPNO        = traces.SAMPNO (+))
           and (samples.ORIGNO    = traces.ORIGNO (+)));

SPOOL OFF;
```

This example from the PetChem Database retrieves flat files of a selection of data from PetChem's SAMPLES, MAJORS and TRACES tables. Note the four outer joins required to ensure that a lack of a data from a sample in one table will not prevent the retrieval of existing data in the other tables.

For many people, a major advantage of this method will be the ability to use their favourite PC editor or word processor to edit long and complex SQL*Plus 'programs'. It is no longer necessary to learn the DG operating system and editors in order to master SQL data manipulations. This method can be used to execute any SQL*Plus commands, including updates and table creations.

6 - SUMMARY

GetSeq follows what is termed 'client-server methodology'. The remote host computer with the database manager is the server that handles the data manipulation, queries and joins. GetSeq is the client program running on a PC that sends SQL commands, receives the the required data from the server and does what it wants with it. In this case that is just to store the data on a DOS file. However, this forms a good basis for further PC client software, particularly programs for generating the specialised scientific plots required in BMR.

Nevertheless, users will find GetSeq useful for rapid retrieval of data from the DG and Oracle to a PC. Of especial value is the ability it gives them to formulate SQL command files on the PC, using their favourite PC editor or word processor. It obviates the need to learn much about the DG operating system, or sluggish shared editors like SED and SLATE.

APPENDIX A - IBM PC CHARACTER SET

0	32	@ 64	` 96	Ç 128	á 160	└ 192	α 224
☺ 1	! 33	A 65	a 97	ü 129	í 161	┴ 193	ß 225
☻ 2	" 34	B 66	b 98	é 130	ó 162	┬ 194	Γ 226
♥ 3	# 35	C 67	c 99	â 131	ú 163	├ 195	π 227
♦ 4	$ 36	D 68	d 100	ä 132	ñ 164	─ 196	Σ 228
♣ 5	% 37	E 69	e 101	à 133	Ñ 165	┼ 197	σ 229
♠ 6	& 38	F 70	f 102	å 134	ª 166	╞ 198	µ 230
• 7	' 39	G 71	g 103	ç 135	º 167	╟ 199	τ 231
◘ 8	(40	H 72	h 104	ê 136	¿ 168	╚ 200	Φ 232
○ 9	) 41	I 73	i 105	ë 137	⌐ 169	╔ 201	Θ 233
◙ 10	* 42	J 74	j 106	è 138	¬ 170	╩ 202	Ω 234
♂ 11	+ 43	K 75	k 107	ï 139	½ 171	╦ 203	δ 235
♀ 12	, 44	L 76	l 108	î 140	¼ 172	╠ 204	∞ 236
♪ 13	- 45	M 77	m 109	ì 141	¡ 173	═ 205	ø 237
♫ 14	. 46	N 78	n 110	Ä 142	« 174	╬ 206	ε 238
☼ 15	/ 47	O 79	o 111	Å 143	» 175	╧ 207	∩ 239
► 16	0 48	P 80	p 112	É 144	▒ 176	╨ 208	≡ 240
◄ 17	1 49	Q 81	q 113	æ 145	▒ 177	╤ 209	± 241
↕ 18	2 50	R 82	r 114	Æ 146	▒ 178	╥ 210	≥ 242
‼ 19	3 51	S 83	s 115	ô 147	│ 179	╙ 211	≤ 243
¶ 20	4 52	T 84	t 116	ö 148	┤ 180	╘ 212	⌠ 244
§ 21	5 53	U 85	u 117	ò 149	╡ 181	╒ 213	⌡ 245
▬ 22	6 54	V 86	v 118	û 150	╢ 182	╓ 214	÷ 246
↨ 23	7 55	W 87	w 119	ù 151	╖ 183	╫ 215	≈ 247
↑ 24	8 56	X 88	x 120	ÿ 152	╕ 184	╪ 216	° 248
↓ 25	9 57	Y 89	y 121	Ö 153	╣ 185	┘ 217	∙ 249
→ 26	: 58	Z 90	z 122	Ü 154	║ 186	┌ 218	· 250
← 27	; 59	[91	{ 123	¢ 155	╗ 187	█ 219	√ 251
∟ 28	< 60	\ 92	| 124	£ 156	╝ 188	▄ 220	ⁿ 252
↔ 29	= 61	] 93	} 125	¥ 157	╜ 189	▌ 221	² 253
▲ 30	> 62	^ 94	~ 126	₧ 158	╛ 190	▐ 222	■ 254
▼ 31	? 63	_ 95	⌂ 127	ƒ 159	┐ 191	▀ 223	255

APPENDIX B - PROGRAM LISTING

```
/* GETSEQ  -  FOR OBTAINING DATA DIRECTLY FROM A REMOTE ORACLE DATABASE */

/* BY R.J. RYBURN */
/* version 1.00, 18-05-90 */
/* Microsoft C version 5.1, Greenleaf Communication library S3-2.10 */


/************************* INCLUDE FILES ******************************/

#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include "gf.h"
#include "ibmkeys.h"
#include "asiports.h"


/************************* NAMED CONSTANTS ****************************/

#define MODE            ASINOUT|ASCII|NORMALRX
#define BUF_LEN         1024
#define WORD_LEN  8


/* THE FOLLOWING ASSEMBLY LANGUAGE FUNCTIONS ARE IN THE FILE SCREEN.AS **

   clear()              - clear whole screen & home cursor
   cursor(line,column)  - move cursor to line and column
   aput(char,attrib)    - write character with attribute to screen
   aputs(char*,attrib)  - write string with attribute to screen


************************* FUNCTION PROTOTYPES ************************/

void clear(), cursor(int,int), aputs(char*,int), readlog(), automatic();
void mmenu(), mputs(int,char*,int), mainmen(), termem(), smenu(), echeck();
void setmen(), loghost(int), comport(), network(), dgcon(), beep();
void capture(int), hangon(), paktp(), putg(char), putgs(char*), finito();
void tocaps(char *), senter(char*,char*,int), computs(char*), putinp(char);
void puty(char*), putsrev(char*), putbs(char*), keycode(char*,char*);
char *gets(), *backstep(char*);
int  wordcomp(char*,char*), atoi(char*), getcom(int), waitfor(char *p);
long time(long *);


/******** FUNCTIONS FROM THE GREENLEAF COMMUNICATIONS LIBARY ************

int asiopen(port,mode,rx_len,tx_len,baud,parity,stopbits,wordlen,dtr,rts)
   MAIN GREENLEAF FUNCTION FOR OPENING A COM PORT AND STARTING INTERRUPTS
                        USES MANY OTHER UNSEEN GREENLEAF COMMS FUNCTIONS
int      asigetc(port)          GET A CHARACTER (AS AN INT) FROM COM PORT
int      asiputc(port,charint)  SENT A CHARACTER (AS INTEGER) TO COM PORT
int      asiputs(port,string,option)          SEND A STRING TO A COM PORT
int      asiquit(port)                    TERMINATE INTERRUPTS ON COM PORT
int      exit(status)      SPECIAL exit() THAT TURNS OFF COMMS INTERRUPTS


************************* GLOBAL VARIABLES ***************************/

char inbuf[2000];                       /* INPUT BUFFER                 */
char outbuf[2000];                      /* OUTPUT BUFFER                */
char logbuf[1000];                      /* BUFFER FOR LOGON/OFF STRING */
char capname[65];                       /* PATHNAME OF CAPTURE FILE     */
int  capfile=0;                         /* HANDLE FOR CAPTURE FILE      */
char capbuf[1030];                      /* CAPTURE ACCUMULATION BUFFER */
char *pcap, *qcap;                      /* POINTERS TO CAPTURE BUFFER   */
char *endlist;                          /* POINTER TO ENDOFLIST STRING */
int  exitflag=0;                        /* SUCCESS = 0, FAILURE > 1     */
int  forever=1;                         /* SOME SYNTACTIC SUGAR         */
int  port=0;                            /* CURRENT PORT (COM1=0, 2=1)   */
int  bauds[] = { 300, 600, 1200, 2400, 4800, 9600, 19200 }; /* RATES     */
int  baud=5;                            /* CURRENT BAUD RATE            */
int  sbits=1;                           /* CURRENT NO. OF STOP BITS     */
int  parity=0;                          /* CURRENT PARITY SELECTION     */
int  xon=1;                             /* FLAG FOR XON/XOFF PROTOCOL   */
int  comcon=0;                          /* FLAG FOR PORT CONNECTION     */
int  hostcon=0;                         /* FLAG FOR DG CONNECTION       */
int  oracon=0;                          /* FLAG FOR ORACLE CONNECTION   */
int  capflag=0;                         /* FLAG FOR FILE CAPTURE        */
```

```
int  timeout=10;                        /* CURRENT TIMEOUT IN SECONDS  */
int  timeout2=10;                       /* TEMP VAR FOR LOGON TIMEOUT  */
char *password1;                        /* POINTER TO DG PASSWORD      */
char *password2;                        /* POINTER TO ORACLE PASSWORD  */
char *source;                           /* POINTER TO COMMAND PARAM 1  */
int  sqlflag=0;                         /* FLAG TO INDICATE SQL SELECT */
char linend[6] = { "\r" };              /* LINE TERMINATOR CODE        */
char backsp[6] = { "\x7F" };            /* BACKSPACE CODE              */
char delete[6] = { "\b" };              /* DELETE KEY CODE             */
char larrow[6] = { "\x19" };            /* LEFT ARROW KEY CODE         */
char rarrow[6] = { "\x18" };            /* RIGHT ARROW KEY CODE        */

/*************************************************************************/


main(argc,argv) /* GET THE LOGON STRING, GET THE COMMAND LINE
                   ARGUMENTS (IF ANY), AND BRANCH ACCORDINGLY */
        int argc; char *argv[]; { char null=0;
        password1=&null; password2=&null;   /* INITIALIZE PASSWORDS TO NULL */
        readlog();                           /* GET THE LOGON/OFF STRING */
        if (argc>4) password2=argv[4];              /* GET ORACLE PASSWORD */
        if (argc>3) password1=argv[3];                /* GET HOST PASSWORD */
        *capname=0;                  /* INITIALIZE CAPTURE FILE NAME TO NULL */
        if (argc>2) strcpy(capname,argv[2]);   /* GET DOS CAPTURE FILE NAME */
        if (argc>1) { source=argv[1];                    /* GET SOURCE STRING */
            automatic(); }                    /* ATEMPT AUTOMATIC TRANSFER */
        readlog(); mainmen(); }

void readlog() /* READ LOGON FILE INTO LOGON BUFFER */
        { int file=open("GSLOG.ON",O_RDONLY);
        if (file>=0) { read(file,logbuf,1000U); close(file); } }

void automatic() /* AUTOMATIC EXTRACTION OF FILE OR SQL SELECT OUTPUT
                    FROM HOST TO DOS CAPTURE FILE */
        { char *q, buff[1024]; int len, flag, file;
        if (*source=='(') { sqlflag=1;       /* BRACKETS AROUND TABLE NAME */
            ++source; q=source; while (*q) ++q; --q; if (*q==')') *q=0; }
        else if (*source=='@') { ++source; sqlflag=2; }     /* DOS SQL FILE */
        else if (!wordcomp(source,"SELECT")) sqlflag=3;
        else if (*source=='?') { *capname=0; sqlflag=4; }    /* PROMPT MODE */
        else loghost(1);
        if (sqlflag) loghost(2);
        if (!hostcon) return;
        capture(0);                  /* OPEN CAPTURE FILE AND SET CAPTURE FLAG */
        if (!capflag) return;
        if (!sqlflag) {              /* RETRIEVE AOS FILE WITH PATHNAME source */
            asiputs(port,"ty ",-1); computs(source); }
        else {
            if (sqlflag==1) { /* SELECT ALL RECORDS FORM GIVEN ORACLE TABLE */
                computs("SET PAGESIZE 50000;");           /* NO PAGE BREAKS */
                asiputs(port,"SELECT * FROM ",-1); asiputs(port,source,-1);
                computs(";"); }
            else if (sqlflag==2 &&      /* GET SQL COMMAND(S) FROM DOS FILE */
                (file=open(source,0))>=0) {      /* OPEN NOMINATED DOS FILE */
                    len=read(file,buff,1024); close(file); /* READ SQL FILE */
                    q=buff+len-1; while (*q==26) --q; *++q=0;   /* NULL END */
                    computs(buff); }                 /* SEND SQL COMMAND TO HOST */
            else if (sqlflag==3) {  /* GET SQL COMMAND FROM FIRST PARAMETER */
                q=source;
                while (*q) { if (*q=='_') *q=' '; ++q; } /* STRIP UNDERLINE */
                computs(source); }                   /* SEND SQL COMMAND TO HOST */
            else if (sqlflag=4) { q=buff; *q=0;  /* PROMPT FOR SQL COMMANDS */
                q=buff; *q=0; putbs("\n");
                while (forever) {
                    if (!*q) putbs("\n\n Enter SQL commands. "
                                   "Press ENTER again when finished");
                    *q=0; senter("\n",q,64); if (!*q) break;
                    while (*q) ++q; --q;
                    if (*q==';') {          /* TRANSMIT SQL COMMAND TO HOST */
                        putbs("\n\n"); q=buff; computs(q);
                        waitfor(endlist); *q=0; putbs("\n\n"); }
                    else { ++q; *q++=' '; *q=1; } }
                finito(); }
            waitfor("-"); }         /* MAKE SURE SQL HEADER HAS BEEN REACHED */
        waitfor(endlist);
        finito(); }

void mmenu() /* DISPLAY MAIN MENU */
        { clear(); cursor(0,25); mputs(12,
```

```
                    "GETSEQ MAIN MENU\n\n\n"


        "1  -  Setup Menu\n\n"

        "2  -  Terminal Mode\n\n"

        "3  -  Automatic logon to host -------\n\n"

        "4  -  Automatic logon to ORACLE------\n\n"

        "5  -  Capture incoming data on ------\n\n"

        "6  -  Log Off from ORACLE and/or Host\n\n"

        "7  -  Exit to DOS\n\n\n"


        "            CHOOSE BY NUMBER",3);

        cursor(26,0); }

void mputs(n,p,a) /* DISPLAY A MENU p INDENTED n COLUMNS, WITH ATTRIBUTE a */
        int n; char *p; int a; { int i;
        while (*p) {
            if(*p=='\n') {
                while (*p=='\n') { aput(*p,7); ++p; }
                for (i=0;i<n;++i) aput(' ',7);
                continue; }
            else { aput(*p,a); ++p; } } }

void mainmen() /* DISPLAY MAIN MENU, INTERCEPT KEYS, & BRANCH ACCORDINGLY */
        { static char *m[] = { "LOGGED OFF", "LOGGED ON " };
        mmenu();
        while (forever) {
            cursor(7,50); aputs(m[hostcon],48);
            cursor(9,50); aputs(m[oracon],48);
            cursor(11,50);
            if (capflag) aputs(capname,48);  /* DISPLAY CAPT' FILE PATHNAME */
            else aputs("CAPTURE OFF",48);     /* ELSE DISPLAY "CAPTURE OFF" */
            aputs("                          ",7); /* BLANK OUT ANY PRIOR DISPLAY */
            cursor(26,0);
            switch(getkey()) {
                case '1': setmen(); mmenu(); break;
                case '2': termem(); mmenu(); break;
                case '3': loghost(1); mmenu(); break;
                case '4': loghost(2); mmenu(); break;
                case '5': capture(1); if (capflag) mmenu(); break;
                case '6': loghost(3); mmenu(); break;
                case '7':
                case 27 : finito(); break;
                default : beep(); } } }

void finito() /* CLOSE CAPTURE FILE, LOG OFF, CLOSE PORT & EXIT PROGRAM */
         { clear();
         if (capflag) { capture(1); close(capfile); }
         if (hostcon) loghost(3);
         if (comcon) asiquit(port);
         exit(exitflag); }

void smenu() /* DISPLAY SETUP MENU MENU */
        { clear(); cursor(0,25); mputs(12,
                    "GETSEQ SETUP MENU\n\n\n"


        "1  -  Communications port -----------\n\n"

        "2  -  Baud Rate ---------------------\n\n"

        "3  -  Stop Bits ---------------------\n\n"

        "4  -  Parity ------------------------\n\n"

        "5  -  X-on/X-off protocol -----------\n\n"

        "6  -  Enter key generates -----------\n\n"

        "7  -  Backspace key generates -------\n\n"

        "8  -  Delete Key generates ----------\n\n"

        "9  -  Cursor left arrow -------------\n\n"

        "A  -  Cursor right arrow ------------\n\n\n"
```

```
        "      CHOOSE BY NUMBER - ESCAPE TO EXIT",3);

        cursor(26,0); }


void setmen() /* DISPLAY SETUP MENU & BRANCH ACCORDINGLY */
        { int oldport=port;
        static char *po[] = { "COM1", "COM2" };
        static char *b[]  = {
            "300","600","1200","2400","4800","9600","19200" };
        static char *s[]  = { "1","2" };
        static char *pa[] = { "NONE","ODD","EVEN","SPACE","MARK" };
        static char *x[]  = { "OFF", "ON" };
        smenu();
        while (forever) {
            cursor(3,50); aputs(po[port],48);
            cursor(5,50); aputs(b[baud],48);    aputs("  ",7);
            cursor(7,50); aputs(s[sbits-1],48);
            cursor(9,50); aputs(pa[parity],48); aputs("  ",7);
            cursor(11,50); aputs(x[xon],48);    aput(' ',7);
            cursor(13,50); aputs(linend,48);    aputs("    ",7);
            cursor(15,50); aputs(backsp,48);    aputs("    ",7);
            cursor(17,50); aputs(delete,48);    aputs("    ",7);
            cursor(19,50); aputs(larrow,48);    aputs("    ",7);
            cursor(21,50); aputs(rarrow,48);    aputs("    ",7);
            cursor(26,0);
            switch(getkey()) {
                case '1': if (port) port=0; else port=1; break;
                case '2': if (baud<6) ++baud; else baud=0; break;
                case '3': if (sbits<2) ++sbits; else sbits=1; break;
                case '4': if (parity<4) ++parity; else parity=0; break;
                case '5': if (xon) xon=0; else xon=1; break;
                case '6': keycode(linend,"ENTER key");      smenu(); break;
                case '7': keycode(backsp,"BACKSPACE key"); smenu(); break;
                case '8': keycode(delete,"DELETE key");     smenu(); break;
                case '9': keycode(larrow,"LEFT ARROW");     smenu(); break;
                case 'A': keycode(rarrow,"RIGHT ARROW");    smenu(); break;
                case 27 :
                case 13 : if (comcon) { asiquit(oldport); comcon=0; }
                          comport(); return; break;
                default : beep(); } } }

void keycode(p,q) /* ENTER THE CODES CORRESPONDING TO KEY q INTO BUFF p */
        char *p, *q; { char buff[65], *r; int i; clear(); r=buff;
        putbs("\n\n  Enter decimal ASCII code(s) for ");
        putbs(q); *buff=0; senter("\n ",buff,32); r=buff;
        while (*p++=atoi(r)) { while (*r==' ') ++r; while (*r>' ') ++r; } }

void comport() /* CONNECT PC TO COM2 OR COM2 */
        { int stat;
        if (!comcon) {
            if ((stat=asiopen(port,MODE,BUF_LEN,BUF_LEN,bauds[baud],parity,
                sbits,WORD_LEN,ON,ON))!=ASSUCCESS) {
                 clear(); puts("SORRY!  CAN'T OPEN COM PORT\n"); exit(1); }
            if (xon) asixon(port,30,70,0,0);   /* TURN ON XON/XOFF PROTOCOL */
            comcon=1; } }

void loghost(n) /* PEFORM AUTOMATIC LOGON TO HOST COMPUTER */
        int n; { char *p=logbuf, *q, input[21]; int i, key=0;
        comport();              /* MAKE SURE COM PORT I/O IS SET UP & RUNNING */
        clear(); putbs("GETSEQ AUTOMATIC LOGON/LOGOFF\n\n");
        if (n==1 && hostcon>0) {
            putbs("ALREADY LOGGED ON TO HOST! "); paktp(); return; }
        if (!*logbuf) {
            putbs("LOGON FILE NOT FOUND!  CAN'T LOG ON"); paktp(); return; }
        timeout=10;                         /* SET DEFAULT TIMEOUT TO 10 SECONDS */
        if (n==2 && hostcon) {    /* ORACLE LOGON WHERE HOST ALREADY LOGGED */
            if (oracon) return; while (*p!='O' || wordcomp(p,"ORACLE")) ++p; }
        if (n==3) { while (*p!='L' || wordcomp(p,"LOGOFF")) ++p; }
        timeout=timeout2;
        while (forever) {
            while (*p>' ') ++p;                     /* FIND NEXT SPACE OR EXIT */
            while (*p==' ' || *p==10 || *p==13) ++p;     /* SKIP SPACES ETC */
            if (!*p || *p==27) break;
            if (*p<='Z') { /* EXIT IF NOT ORACLE LOGON FOLLOWING HOST LOGON */
                if (n==2 && !wordcomp(p,"ORACLE")) continue; else break; }
            if (*p=='t') {
                ++p; timeout=atoi(p);               /* SET TIMEOUT TO p SECS */
                putbs("TIMEOUT SET TO "); for (q=p;*q>' ';++q) aput(*q,3);
                putbs(" SECONDS\n\n"); }
            if (*p=='w') { ++p; hangon(atoi(p)); }    /* WAIT FOR p SECONDS */
            else if (*p=='r') {                     /* PROCESS "RECEIVE" STRING */
                q=p+=2; while (*p && *p!='"') ++p; *p=0;
                endlist=q;                  /* GET POINTER TO RECEIVE STRING */
                i=waitfor(q); if(!i) return;            /* TIMEOUT OCCURRED */
                ++p; }
```

```
            else if (*p=='s') { ++p;                    /* PROCESS "SEND" STRING */
                while (forever) {
                    if (*p=='"') {  /* SEND STRING DEFINED BY DOUBLE QUOTES */
                        ++p; while (*p && *p!='"') asiputc(port,*p++); ++p; }
                    else {                  /* SEND ASCII CHAR DEFINED BY NUMBER */
                        asiputc(port,atoi(p));              /* SEND CHARACTER */
                        while (*p>='0' && *p<='9') ++p; }  /* END OF NUMBER */


                    if (*p<=' ') break;
                    if (*p==',') ++p; } }
            else if (*p=='p') { ++p;          /* ENTER AND TRANSMIT PASSWORD */
                              /* TRANSMIT PASWORDS SUPPLIED ON COMMAND-LINE */
                if (*p=='1' && *password1) computs(password1);
                else if (*p=='2' && *password2) computs(password2);
                else {              /* OTHERWISE ENTER PASSWORD INTERACTIVELY */
                    putbs("\n\nI mean really enter your password! : ");
                    while (key!=13) {
                        if (kbhit()) asiputc(port,(key=getkey())); }
                    putbs("\n"); }
                key=0; } }
        hostcon=1;
        if (n==2) oracon=1;
        else if (n==3) hostcon=oracon=0; timeout2=timeout; timeout=32000;}

int waitfor(p) /* WAIT FOR STRING p IN INCOMMING TEXT FROM HOST COMPUTER */
               /* RETURN 1 IF FOUND, 0 IF TIMED OUT */
        char *p; { int flag=0, tflag=0; char c, *q=p; long tend;
        while (forever) {
            if (kbhit()) {                              /* CHECK FOR KEYSTROKE */
                if (getkey()==CTRLEND) {    /* INTERCEPT CONTROL-END POSING */
                    asiputc(port,3); asiputc(port,1); } }     /* FOR CTRL-C */
            else if ((c=getcom(0))>=0) {
                putg(c); tflag=0;
                if (c==*q) {        /* LOOK FOR END STRING.  RETURN IF FOUND */
                    ++q; flag=1;
                    if (!*q) { echeck(); return(1); } } /* CHECK FOR ERRORS */
                else { flag=0; q=p; } }
            else { /* EXIT IF NO CHARACTERS RECEIVED WITHIN TIMEOUT SECONDS */
                if (!tflag) { tend=time(NULL)+timeout; tflag=1; }
                else if (time(NULL) > tend) {
                    putbs("\n\nTIMEOUT OCCURRED!  ABORTING!");
                    paktp(); return(0); } } } }

void echeck() /* CHECK FOR ERRORS IN RETRIEVING DATA */
        { char *p=pcap; p=backstep(p)-1; p=backstep(p)+1;
        if (!wordcomp(p,"Warning: File does not exist,")) exitflag=1;
        else { p-=2; p=backstep(p)+1;
            if (!wordcomp(p,"no records selected,")) exitflag=2;
            else if (!wordcomp(p,"ERROR at line")) exitflag=3; } }

char *backstep(p) char *p; { while (p>capbuf && *p!=10) --p; return(p); }

void termem() /* TERMINAL EMULATOR.  KEY CODES ARE DEFINED IN ibmkeys.h */
        { int stat, c; unsigned key; clear();
          putgs("TERMINAL EMULATION - PRESS ESCAPE TO EXIT\n\n");
          if (!comcon) comport();
        while (forever) {
             if((c=getcom(0))>=0) {
             if (c==25) putch(8);   /* TRANSLATE ASCII 25 TO CURSOR LEFT */
             else if (c==12) clear();     /* ASCII 12 MEANS CLEAR SCREEN */
                else if (c) putg(c); }
            if(kbhit()) {                               /* CHECK FOR KEYSTROKE */
                key=getkey();                                 /* GET KEYSTROKE */
                switch(key) {  /* INTERCEPT CRITICAL KEYSTROKES & TRANSLATE */
                    case       ESC: return;
                    case   CTRLEND: asiputc(port,3);     /* USE CONTROL-END */
                                    asiputc(port,1); break;   /* FOR CTRL-C */
                    case    DELETE: asiputs(port,delete,-1); break;
                    case     CURLF: asiputs(port,larrow,-1); break;
                    case     CURRT: asiputs(port,rarrow,-1); break;
                    case        13: asiputs(port,linend,-1); break;  /* ENT */
                    case         8: asiputs(port,backsp,-1); break;  /* BSP */
                    default       :
                        if (key<256) asiputc(port,(int)key); } } } }

void capture(flag) /* OPEN FILE TO CAPTURE INCOMMING DATA, OTHERWISE CLOSE */
        int flag; { int len; char *q, yes[2];
        if (capflag) {          /* CLOSE CAPTURE FILE AND SET capflag TO ZERO */
            getcom(1); write (capfile,"\x1A\x1A\x1A\x1A",4);   /* 4 CTRL-Zs */
            close(capfile); capflag=0; return; }
        if (flag) { clear(); cursor(0,20);
            aputs("CAPTURE INCOMMING DATA ON A FILE",3); cursor(4,0); }
        while (forever) {
            if (flag || *capname==0) { senter(
                "\n\n Enter DOS pathname for capture file, or ESC to exit\n",
                capname,60); tocaps(capname); }
```

```
            if (!*capname) return;
            if ((capfile=open(capname,O_WRONLY))>0) { /* SEE IF FILE EXISTS */
                *yes='N'; yes[1]=0; close(capfile);
                senter("\n\n File already exists!  OK to overwrite? (Y/N)",
                    yes,1); putgs("\n");
                if (*yes=='Y' || *yes=='y') {   /* REOPEN FILE AND TRUNCATE */
                    capfile=open(capname,O_WRONLY|O_TRUNC); break; }
                else { flag=1; continue; }}
            if ((capfile=creat(capname,S_IWRITE))>=0) break; /* CREATE FILE */
            putbs("SORRY!  CAN'T OPEN FILE!"); paktp();
            *capname=0; continue; }
        for (q=endlist;*q;++q); len=q-endlist;  /* GET PROMPT STRING LENGTH */
        write(capfile,endlist,len);  /* WRITE PROMPT STRING TO CAPTURE FILE */
        capflag=1; pcap=capbuf; qcap=pcap+1024; } /* SET CAP BUFFER POINTRS */

int getcom(flag)  /* GET COM CHAR. WRITE BUFFER TO DISK IF FULL OR FLAG SET */
        int flag; { int c; unsigned int len; char *p, *q; c=-1;
        if (!isrxempty(port)) {
            c=asigetc(port);
            if (!capflag) return(c);
            else { *pcap=c; ++pcap; } }
        else if (!flag) return(c);
         if (pcap>=qcap || flag) {
             if (flag) len=pcap-capbuf;
             else len=512;
             if (write(capfile,capbuf,len)<0) {
                 putgs("\n\nERROR WRITING TO CAPTURE FILE!"); paktp();
                 capflag=0; }
             else {
                  for (p=capbuf,q=capbuf+len;q<pcap;++p,++q) *p=*q;
                pcap=p; } }
        return(c); }

void hangon(n) /* WAIT AND DO NOTHING FOR n SECONDS */
        int n; { long tend=time(NULL)+n; while (time(NULL) < tend); }

void beep() { putch(7); }

void paktp() /* PRESS ANY KEY TO PROCEED - WAIT FOR KEYSTROKE */
        { putbs(" -- press any key to proceed "); getkey(); aput(10,7); }

void putg(c) /* PUT GREEN CHARACTER WITH SOME CHECKS */
        char c; { if (c==7) beep; else if (c!=13) aput(c,2); }

void putgs(p) /* PUT GREEN STRING p WITH SAME CHECKS AS putg() */
        char *p; {
        while (*p) { if (*p==7) beep; else if (*p!=13) aput(*p,2); ++p; } }

void putbs(p) /* PUT BLUE STRING p WITH SAME CHECKS AS putg() */
        char *p; {
        while (*p) { if (*p==7) beep; else if (*p!=13) aput(*p,3); ++p; } }

void computs(p) /* PUT STRING p TO COM PORT port, APPEND linend TERMINATOR */
        char *p; { asiputs(port,p,-1); asiputs(port,linend,-1); }

int atoi(p) /* REPLACES STANDARD LIBRARY ASCII-TO-INTEGER FUNCTION */
        char *p; { int i=0, sign=0; while (*p==' ') ++p;
        if (*p=='-') { sign=1; ++p; }  else if (*p=='+') ++p;
        while (*p>='0' && *p<='9') { i*=10; i+=*p-'0'; ++p; }
        if (sign) i=-i; return(i); }

int wordcomp(p,q) /* COMPARE STRING p WITH q.  RETURN 0 IF STRINGS COMPARE */
        char *p, *q; { while (*p==*q) { ++p; ++q; }
        if (!*q) return(0); else return(1); }

void tocaps(p) /* CONVERT STRING p TO UPPER CASE */
        char *p; { while (*p) { if (*p>='a' && *p<='z') *p-=32; ++p; } }

 /* STRING INPUT WITH DISPLAYED PROMPT & UNDERLINED INPUT FIELD OF DEFINED
    LENGTH.  THE DEFAULT INPUT (IF ANY) IS INITIALLY DISPLAYED IN THE INPUT
    FIELD.  INPUT IS TERMINATED BY A RETURN.  FULL EDITING FUNCTIONS
    - INCLUDING CHARACTER INSERT - APPLY WITHIN THE INPUT FIELD.
    IF len IS NEGATIVE PROMPT IS DISPLAYED IN INVERSE VIDEO */

void senter(prompt,input,len) char *prompt, *input; int len; {
        char c, *p=input, *q, *r; int ins=0;
        puty(prompt); puty(" : "); q=p+len-1;
        while (p<=q && *p) putinp(*(p++));
        while (p<=q) putinp(*(p++)=' ');
        while (p>input) { p--; putch(8); }
        while ((c=getch())!=13 && c!=27) {      /* ESCAPE OR RETURN TO EXIT */
          if (c>31 && c<127) { putinp(c);
            if (ins && p<q) { r=q; while (r>p) { *r=*(r-1); --r; } ++r;
              while (r<=q) putinp(*(r++)); --r;
              while (r>p) { putch(8); r--; } }
            *p=c; if (p<q) p++; else { putch(8); } }
```

```
          else { if (!c) c=getch();
            switch (c) {
              case 3        : clear(); exit(1);
              case 19       :
              case 75       : if (p>input) { putch(8); p--; } break; /* LEFT */
              case 4        :
              case 77       : if (p<q) putinp(*(p++)); break; /* RIGHT ARROW */
/*BACKSPACE*/ case 8        : if (p>input) { putch(8); p--; } else break;
              case 7        :
              case 83       : for (r=p;r<q;r++) putinp(*r=*(r+1)); /* DELETE */
                  putinp (*r=' '); r++; while (r>p) { putch(8); r--; } break;
              case 25       : while (p>input) { putch(8); p--; }
                  while (p<=q) { putinp (*p=' '); p++; }
                  while (p>input) { putch(8); p--; } break;
              case 22       :
              case 82       : for (r=p;r>=input-1;r--) putch(8);   /* INSERT */
                  if (ins) { ins=0; putch(':'); }
                  else { ins=1; putch('^'); }
                  putch(' '); ++r; while (++r<p) putinp(*r); break; } } }
        if (c==27) *input=0;
        else { while (*q==' ' && q>=input) q--; *(q+1)=0;
            if (*input<=' ' && *input>0) {     /* REMOVE ANY LEADING SPACES */
              for(p=input;*p<=' ';++p); for(q=input;*p;) *q++=*p++; *q=0; } }
        puty("\n"); }

void putinp(c)  /* PUTCHAR WITH UNDERLINES REPLACING ASCII SPACES */
        char c; { aput(c,78); }

void puty(p) /* REPLACES THE STANDARD puts() LIBRAY FUNCTION BUT NO CRLF AT END */
        char *p; { while(*p) { aput(*p,3); ++p; } }

void putsrev(p) /* DISPLAY STRING IN INVERSE VIDEO */
        char *p; { while(*p) { aput(*p,48); ++p; } }
```