```
 1
 2 /*********************************************/
 3 /*                                           */
 4 /* routines this module lcd_3 -              */
 5 /* vs 07/04/2013 at 08:00                    */
 6 /*                                           */
 7 /* 0) void setup                             */
 8 /* 1) void loop                              */
 9 /* 1) begin_ram - init ram memory check      */
10 /* 2) int freeram - return available ram space  */
11 /* 3) get_stars - read star data from eprom     */
12 /* 4) gettgt_1 - select a calibration star      */
13 /* 2) dgrepos - get axis degree position        */
14 /* 2) mresolve_1 - get axes tiks/rev menu       */
15 /* 3) mbklash_1 - get axes backlash measurement */
16 /* 1) corpos - get backlash corrected position  */
17 /* 2) menugo - operate a 4 line menu/options    */
18 /* 3) rabl_acw - get single axis backlash parms */
19 /* 2) byte getkbd - read buttons w. s/w debounce*/
20 /* 2) byte getkbd2 - read keyboard buttons      */
21 /* 2) getpos - return decimal cnt of a channel  */
22 /* 1) makbin - convert 8 chars to a long int    */
23 /* 2) suckc - wait for a char on a serial line  */
24 /* 2) lcd_out_msg - write a string              */
25 /* 2) mak_lcd_lut - make lcd lookup table       */
26 /* 3) setrowcol - set DDRAM address to row,col  */
27 /* 3) init_lcd - set up lcd                     */
28 /* 4) clrlcd - clear LCD display                */
29 /* 4) wbytir - write byte to instruction reg    */
30 /* 5) wnibir - write nibble to instruction reg  */
31 /* 4) wbytdr - write byte to data reg           */
32 /* 5) wnibdr - write nibble to data reg         */
33 /* 6) cycle_ena_latch - cycle lcd enable latch  */
34 /* 7) select_ireg - select lcd instruction reg  */
35 /* 8) select dreg - select lcd data reg         */
36 /* 9) select write - select write to lcd        */
37 /* 0) select_read - select read from lcd        */
38 /*                                           */
39 /*********************************************/
40
41 /*********************************************/
42 /*                                           */
43 /* direct Port i/o                           */
44 /* PORTB bits to arduino pin #:              */
45 /*  0 1  2  3  4  5  6 7 BIT#                */
46 /*  8 9 10 11 12 13 (xtl6,xtl7) -ino dig pin#   */
47 /*                                           */
48 /* PORTB to arduino digital pin masks:       */
49 /* Pin 8 = 0x01                              */
50 /* Pin 9 = 0x02                              */
51 /* Pin10 = 0x04                              */
52 /* Pin11 = 0x08                              */
53 /* Pin12 = 0x10                              */
54 /* Pin13 = 0x20                              */
55 /*                                           */
56 /* PORTD maps to arduino dig pins 0 to 7     */
57 /* PORTD masks:                              */
58 /*  Pin 0 = 0x01                             */
59 /*  Pin 1 = 0x02                             */
60 /*  Pin 2 = 0x04                             */
61 /*  Pin 3 = 0x08                             */
62 /*  pin 4 = 0x10                             */
63 /*  pin 5 = 0x20                             */
64 /*  pin 6 = 0x40                             */
65 /*  pin 7 = 0x80                             */
66 /*                                           */
67 /* PORTC maps to anlogue pins A0-A5 or digital  */
68 /* pins 14-19                                */
69 /* Pin 14 = 0x01, analogue A0                */
70 /* Pin 15 = 0x02, A1                         */
71 /* Pin 16 = 0x04, A2                         */
72 /*                                           */
73 /*********************************************/
```



```
74
75 /* define bright star database in program memory */
76 #include <avr/pgmspace.h>
77
78 /* define constellations */
79 prog_char cnstell_00[] PROGMEM = "Andromeda";
80 prog_char cnstell_01[] PROGMEM = "Aquila";
81 prog_char cnstell_02[] PROGMEM = "Auriga";
82 prog_char cnstell_03[] PROGMEM = "Bootes";
83 prog_char cnstell_04[] PROGMEM = "Canis Major";
84 prog_char cnstell_05[] PROGMEM = "Canis Minor";
85 prog_char cnstell_06[] PROGMEM = "Cassiopeia";
86 prog_char cnstell_07[] PROGMEM = "Corona Borealis";
87 prog_char cnstell_08[] PROGMEM = "Gemini";
88 prog_char cnstell_09[] PROGMEM = "Gemini";
89 prog_char cnstell_10[] PROGMEM = "Cygnus";
90 prog_char cnstell_11[] PROGMEM = "Leo";
91 prog_char cnstell_12[] PROGMEM = "Leo";
92 prog_char cnstell_13[] PROGMEM = "Lyra";
93 prog_char cnstell_14[] PROGMEM = "Orion";
94 prog_char cnstell_15[] PROGMEM = "Orion";
95 prog_char cnstell_16[] PROGMEM = "Perseus";
96 prog_char cnstell_17[] PROGMEM = "Pegasus";
97 prog_char cnstell_18[] PROGMEM = "Scorpio";
98 prog_char cnstell_19[] PROGMEM = "Taurus";
99 prog_char cnstell_20[] PROGMEM = "Ursa Major";
100 prog_char cnstell_21[] PROGMEM = "Ursa Major";
101 prog_char cnstell_22[] PROGMEM = "Ursa Minor";
102 prog_char cnstell_23[] PROGMEM = "Virgo";
103
104 /* define constellation string addresses */
105 PROGMEM const prog_char *cnstell_addr[] = {cnstell_00,cnstell_01,cnstell_02,cnstell_03,cnstell_04,
        cnstell_05,
106 cnstell_06,cnstell_07,cnstell_08,cnstell_09,cnstell_10,cnstell_11,cnstell_12,cnstell_13,cnstell_14,
107 cnstell_15,cnstell_16,cnstell_17,cnstell_18,cnstell_19,cnstell_20,cnstell_21,cnstell_22,cnstell_23};
108
109 /* define star names */
110 prog_char stname_00[] PROGMEM = "Alpheratz a And";
111 prog_char stname_01[] PROGMEM = "Altair a Aql";
112 prog_char stname_02[] PROGMEM = "Capella a Aur";
113 prog_char stname_03[] PROGMEM = "Arcturus a Boo";
114 prog_char stname_04[] PROGMEM = "Sirius a CMa";
115 prog_char stname_05[] PROGMEM = "Procyon a CMi";
116 prog_char stname_06[] PROGMEM = "Schedar a Cas";
117 prog_char stname_07[] PROGMEM = "Alphecca a CrB";
118 prog_char stname_08[] PROGMEM = "Castor a Gem";
119 prog_char stname_09[] PROGMEM = "Pollux b Gem";
120 prog_char stname_10[] PROGMEM = "Deneb a Cyg";
121 prog_char stname_11[] PROGMEM = "Regulus a Leo";
122 prog_char stname_12[] PROGMEM = "Denebola b Leo";
123 prog_char stname_13[] PROGMEM = "Vega a Lyr";
124 prog_char stname_14[] PROGMEM = "Betelgeuse a Ori";
125 prog_char stname_15[] PROGMEM = "Rigel b Ori";
126 prog_char stname_16[] PROGMEM = "Mirfak a Per";
127 prog_char stname_17[] PROGMEM = "Scheat b Peg";
128 prog_char stname_18[] PROGMEM = "Antares a Sco";
129 prog_char stname_19[] PROGMEM = "Aldebaran a Tau";
130 prog_char stname_20[] PROGMEM = "Dubhe a UMa";
131 prog_char stname_21[] PROGMEM = "Alkaid e UMa";
132 prog_char stname_22[] PROGMEM = "Polaris UMi";
133 prog_char stname_23[] PROGMEM = "Spica a Vir";
134
135 /* define star name string addresses */
136 PROGMEM const prog_char *stname_addr[] = {stname_00,stname_01,stname_02,stname_03,stname_04,stname_05,
137 stname_06,stname_07,stname_08,stname_09,stname_10,stname_11,stname_12,stname_13,stname_14,
138 stname_15,stname_16,stname_17,stname_18,stname_19,stname_20,stname_21,stname_22,stname_23};
139
140 /* define star J2000.0 coordinates RA hrs, mins sec, DEC degrees, mins, secs */
141 PROGMEM prog_int16_t stcoord_00[] = {0, 8, 23, 29, 5, 26}; /* Alpheratz*/
142 PROGMEM prog_int16_t stcoord_01[] = {19, 50, 47, 8, 52, 06}; /* Aquila */
143 PROGMEM prog_int16_t stcoord_02[] = {5, 16, 41, 45, 59, 53}; /* Capella */
144 PROGMEM prog_int16_t stcoord_03[] = {14, 15, 40, 19, 10, 57}; /* Arcturus */
145 PROGMEM prog_int16_t stcoord_04[] = {6, 45, 9, -16, 42, 58}; /* Sirius */
```

```
146 PROGMEM prog_int16_t stcoord_05[] = {7, 39, 18, 5, 13, 30}; /* Procyon */
147 PROGMEM prog_int16_t stcoord_06[] = {0, 40, 31, 56, 32, 14}; /* Schedar */
148 PROGMEM prog_int16_t stcoord_07[] = {15, 34, 41, 26, 42, 53}; /* Alphecca */
149 PROGMEM prog_int16_t stcoord_08[] = {7, 34, 36, 31, 53, 18}; /* Castor */
150 PROGMEM prog_int16_t stcoord_09[] = {7, 45, 19, 28, 1, 34}; /* Pollux */
151 PROGMEM prog_int16_t stcoord_10[] = {20, 41, 26, 45, 16, 49}; /* Deneb */
152 PROGMEM prog_int16_t stcoord_11[] = {10, 8, 22, 11, 58, 02}; /* Regulus */
153 PROGMEM prog_int16_t stcoord_12[] = {11, 49, 4, 14, 34, 19}; /* Denebola */
154 PROGMEM prog_int16_t stcoord_13[] = {18, 36, 56, 38, 47, 01}; /* Vega */
155 PROGMEM prog_int16_t stcoord_14[] = {5, 55, 10, 7, 24, 25}; /* Betelgeuse */
156 PROGMEM prog_int16_t stcoord_15[] = {5, 14, 32, -8, 12, 06}; /* Rigel */
157 PROGMEM prog_int16_t stcoord_16[] = {3, 24, 19, 49, 51, 40}; /* Mirfak */
158 PROGMEM prog_int16_t stcoord_17[] = {23, 3, 47, 28, 4, 58}; /* Scheat */
159 PROGMEM prog_int16_t stcoord_18[] = {16, 29, 24, -26, 25, 55}; /* Antares */
160 PROGMEM prog_int16_t stcoord_19[] = {4, 35, 55, 16, 30, 33}; /* Aldeberan */
161 PROGMEM prog_int16_t stcoord_20[] = {11, 3, 44, 61, 45, 3}; /* Dubhe */
162 PROGMEM prog_int16_t stcoord_21[] = {13, 47, 32, 49, 18, 48}; /* Alkaid */
163 PROGMEM prog_int16_t stcoord_22[] = {2, 31, 49, 89, 15, 51}; /* Polaris */
164 PROGMEM prog_int16_t stcoord_23[] = {13, 25, 12, -11, 9, 41}; /* Spica */
165
166 /* define star co-ord addresses */
167 PROGMEM const prog_int16_t *stcoord_addr[] = {stcoord_00,stcoord_01,stcoord_02,stcoord_03,stcoord_04,
        stcoord_05,
168 stcoord_06,stcoord_07,stcoord_08,stcoord_09,stcoord_10,stcoord_11,stcoord_12,stcoord_13,stcoord_14,
169 stcoord_15,stcoord_16,stcoord_17,stcoord_18,stcoord_19,stcoord_20,stcoord_21,stcoord_22,stcoord_23};
170
171 /* define d/base size and addresses */
172 #define NSTARS 24
173 prog_int16_t *coord_ptr;
174
175 /*********************************************/
176 /* Serial input pins to arduino              */
177 /*                                           */
178 /* Encoder chan 1 (RA) = pin 10              */
179 /* Encoder chan 2 (DEC) = pin 9              */
180 /*                                           */
181 /* Assume on PORTB at 9600 baud              */
182 /*                                           */
183 /*********************************************/
184
185 /* define memory free space counter */
186 #define RAMWRNG 0x0200 /* max space = 0x07ff */
187 int memchk;
188 int startram;
189
190 /* define debug event signal pin */
191 int sigout = 8; /*PortB */
192 int setsig = 0x01;
193 int clrsig = ~0x01;
194
195 /* define serial i/p lines - PortB */
196 #define ISRAMSK  0x04
197 #define ISDECMSK  0x02
198 int serial1 = 10;
199 int serial2 = 9;
200 byte serial1_msk = ISRAMSK;
201 byte serial2_msk = ISDECMSK;
202
203 /* define 9600 baud us time delays */
204 #define DLA9600 100 /* 104 us delay - 9600 baud = 104us bit width */
205 #define DLA9600_2 50
206
207 /* define maximum wait for serial start bit - us */
208 #define MAX_SER_WAIT 9900
209
210 /*********************************************/
211 /* Keyboard input pins to arduino            */
212 /*                                           */
213 /* Up = Pin 14, PortC, 0x01                  */
214 /* Down - Pin 7, PortD, 0x80                 */
215 /* Left - Pin 6, PortD, 0x40                 */
216 /* Right - Pin 15, PortC, 0x02               */
217 /* Enter - Pin 16, PortC, 0x04               */
```



```
218 /*                                           */
219 /*                                           */
220 /*********************************************/
221
222 /* define keyboard lines */
223 int key_up = 14;
224 byte key_up_msk = 0x01;
225 byte ISKUP = 0x01;
226
227 int key_dwn = 7;
228 byte key_dwn_msk = 0x80;
229 byte ISKDWN = 0x02;
230
231 int key_lft = 6;
232 byte key_lft_msk = 0x40;
233 byte ISKLFT = 0x04;
234
235 int key_rght = 15;
236 byte key_rght_msk = 0x02;
237 byte ISKRGHT = 0x08;
238
239 int key_ntr = 16;
240 byte key_ntr_msk = 0x04;
241 byte ISKNTR = 0x10;
242
243 /* null key */
244 byte ISKNULL = 0x80;
245
246 /* define keyboard delays */
247 #define DEBOUNCE 1 /* 20 msec */
248 #define HUMAN 500 /* 2 seconds to read screen */
249
250 /*********************************************/
251 /* LCD to arduino pin outs                   */
252 /*                                           */
253 /*  LCD RS = pin 13                          */
254 /*  LCD RW = pin 12                          */
255 /*  LCD ENA = pin 11                         */
256 /*  LCD D4 = pin 5                           */
257 /*  LCD D5 = pin 4                           */
258 /*  LCD D6 = pin 3                           */
259 /*  LCD D7 = pin 2                           */
260 /*                                           */
261 /*********************************************/
262
263 /* define LCD control lines - PortB */
264 int lcd_rs=13;
265 int lcd_rw=12;
266 int lcd_ena=11;
267
268 int lcd_rs_msk=0x20;
269 int lcd_rw_msk=0x10;
270 int lcd_ena_msk=0x08;
271
272 /* define LCD data bus lines - Port D */
273 int lcd_d4 = 5;
274 int lcd_d5= 4;
275 int lcd_d6=3;
276 int lcd_d7=2;
277
278 int lcd_d4_msk=0x20;
279 int lcd_d5_msk=0x10;
280 int lcd_d6_msk=0x08;
281 int lcd_d7_msk=0x04;
282 byte lcd_xclear; /* nibble to clear lcd data bus */
283
284 /* define lcd data xlate lookup table */
285 byte lcd_lut[16];
286
287 /* define o/p string for LCD */
288 char outstr[81];
289
290 /* define working global variables */
```

```
/* RA and DEC tik counts and direction indicators */
long int rapos;
long int decpos;
int radir;
int decdir;
const char postxt[] = "POS";
const char negtxt[] = "NEG";
char *ra_dirtxt, *dec_dirtxt;

/* axis backlash variables, fwd and rev */
#define RA_BL_FWD 75 /* Pos to neg direction: N(measure) > N(true), so Nt = Nm - (+ve correction) */
#define RA_BL_REV -75 /* Neg to pos direction: Nm < Nt, so Nt = Nm - (-ve correction) */
#define DEC_BL_FWD 75
#define DEC_BL_REV -75
int ra_fwdbl, ra_revbl;
int dec_fwdbl, dec_revbl;

/* backlash corrected counts and flags */
long int cor_rapos, sav_rapos;
long int cor_decpos, sav_decpos;
int cor_radir;
int cor_decdir;

/* define ticks per resolution */
#define EDGES 4
#define TEETH 48
#define RATIO 64
#define SECND 40
#define PRIMR 58
long int RA_RES;
long int DEC_RES;
long int ra_resolve;
long int dec_resolve;
double ra_degrees;
double dec_degrees;
int ra_hrs;
int ra_mins;
int ra_secs;
int dec_deg;
int dec_mins;
int dec_secs;

/* define float o/p string length */
#define DEGLEN 10
char ra_degtxt[DEGLEN+1];
char dec_degtxt[DEGLEN+1];

/* define calibration values */
int align_is_set; /* value >0 if align procedure completed */
int align_star_index; /* index to database of ref stars */

/* save system time at alignment */
unsigned long int align_systime;

/* save aligned axis offsets - degrees */
double align_ra_offst;
double align_dec_offst;
int dec_is_2ndhs; /* flag if DEC in primary hemisphere */

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

void setup()
{
    /*********************************************/
    /*                                           */
    /* setup - initialize pins as outputs etc    */
    /*                                           */
    /*********************************************/

    /* define autos */

```



```
     /* set up debug signal line */
    pinMode(sigout,OUTPUT);
    digitalWrite(sigout,HIGH);

    /* setup keyboard lines as input */
    pinMode(key_up, INPUT);
    pinMode(key_dwn, INPUT);
    pinMode(key_lft, INPUT);
    pinMode(key_rght, INPUT);
    pinMode(key_ntr, INPUT);

    /* setup serial lines as input */
    pinMode(serial1, INPUT);
    pinMode(serial2, INPUT);

    /* set LCDd control lin as o/p */
    pinMode(lcd_rs,OUTPUT);
    digitalWrite(lcd_rs,LOW);

    pinMode(lcd_rw,OUTPUT);
    digitalWrite(lcd_rw,LOW);

    pinMode(lcd_ena,OUTPUT);
    digitalWrite(lcd_ena,LOW);

    /* set LCD data bus lines as o/p */
    pinMode(lcd_d4,OUTPUT);
    digitalWrite(lcd_d4,LOW);

    pinMode(lcd_d5,OUTPUT);
    digitalWrite(lcd_d5,LOW);

    pinMode(lcd_d6,OUTPUT);
    digitalWrite(lcd_d6,LOW);

    pinMode(lcd_d7,OUTPUT);
    digitalWrite(lcd_d7,LOW);

    /*now create lcd LUT to convert data nibble to lcd bus pattern */
    mak_lcd_lut(lcd_lut);

    /* now initialise the lcd */
    init_lcd();

    /* write string to lcd */
    clrlcd();
    lcd_out_msg(0,0,"hello Vivienne!");

    /* init rotation indicators */
    radir = 1;
    decdir = 1;
    cor_radir=1;
    cor_decdir=1;
    ra_dirtxt = (char *) postxt;
    dec_dirtxt = (char *) postxt;
    cor_rapos=0;
    sav_rapos=0;
    cor_decpos=0;
    cor_decpos=0;

    /* load default backlash settings */
    ra_fwdbl = RA_BL_FWD;
    ra_revbl = RA_BL_REV;
    dec_fwdbl = DEC_BL_FWD;
    dec_revbl = DEC_BL_REV;

    /* load default axis resolution settings */
    RA_RES = (long) EDGES;
    RA_RES *= (long) TEETH;
    RA_RES *= (long) RATIO;
    RA_RES *= (long) PRIMR;
    RA_RES /= (long) SECND;
    DEC_RES=RA_RES;
```

```
437     ra_resolve = RA_RES;
438     dec_resolve = DEC_RES;
439     ra_degrees=0;
440     dec_degrees=0;
441
442     /* signal align is not set */
443     align_is_set = -1;
444     dec_is_2ndhs = -1;
445
446     /* wait a bit */
447     delay(1000);
448 }
449
450 /*********************************************/
451 /*                                           */
452 /*********************************************/
453
454 void loop()
455 {
456     /*********************************************/
457     /*                                           */
458     /* loop - loop                               */
459     /*                                           */
460     /*********************************************/
461
462     /* declare autos */
463     int retc;
464
465     /* mem check init */
466     begin_ram();
467
468     /* clear the lcd */
469     clrlcd();
470
471     /* loop forever */
472     for(;;)
473     {
474         /* run backlash menu */
475         mbklash_1(&retc);
476
477         /* get number of ticks per revolution */
478         mresolve_1(&retc);
479
480         /* report position */
481         clrlcd();
482         delay(HUMAN);
483         for (;;)
484         {
485             /* get RA position */
486             dgrpos(serial1_msk,&rapos,&radir,&ra_dirtxt,
487                 &cor_rapos,&sav_rapos,&cor_radir,
488                 ra_fwdbl,ra_revbl,&ra_degrees, ra_resolve,ra_degtxt,&retc);
489
490             sprintf(outstr,"RA = %s %s",ra_degtxt,ra_dirtxt);
491             lcd_out_msg(0,0,outstr);
492
493             sprintf(outstr,"RA = %2.1ihr %2.1imn %2.1is",ra_hrs,ra_mins,ra_secs);
494             lcd_out_msg(2,0,outstr);
495             setrowcol(0,0);
496
497             /* get DEC position */
498             //getpos(serial2_msk,&decpos,&decdir,&dec_dirtxt,&retc);
499             //corpos(serial2_msk,&decpos,&decdir,&dec_dirtxt,
500                 //&cor_decpos,&sav_decpos,&cor_decdir,
501                 //dec_fwdbl,dec_revbl,&retc);
502             //sprintf(outstr,"DEC = %-10.1li %s",cor_decpos,dec_dirtxt);
503
504             /* get DEC position */
505             dgrpos(serial2_msk,&decpos,&decdir,&dec_dirtxt,
506                 &cor_decpos,&sav_decpos,&cor_decdir,
507                 dec_fwdbl,dec_revbl,&dec_degrees, dec_resolve,dec_degtxt,&retc);
508             sprintf(outstr,"DEC = %s %s",dec_degtxt,dec_dirtxt);
509             lcd_out_msg(1,0,outstr);
```



```
510
511             sprintf(outstr,"DEC = %2.1idg %2.1imn %2.1is",dec_deg,abs(dec_mins),abs(dec_secs));
512             lcd_out_msg(3,0,outstr);
513             setrowcol(0,0);
514
515             /* read keyboard */
516             retc = (int) getkbd();
517
518             /* skip out on a key */
519             if(retc != 0) break;
520         } /* end inner loop forever */
521
522         /* get a calibration star */
523         gettgt_1(&retc);
524
525         /* report stack ptr */
526         clrlcd();
527         sprintf(outstr,"RAM free = 0x%04x",memchk);
528         lcd_out_msg(3,0,outstr);
529         setrowcol(3,0);
530         delay(HUMAN);
531
532         /* wait for a key */
533         for (;;)
534         {
535             /* read keyboard */
536             retc = (int) getkbd();
537
538             /* skip out on a key */
539             if(retc != 0) break;
540         } /* end inner loop forever */
541
542     } /* end loop forever */
543
544 } /* end main */
545
546 /*********************************************/
547 /*                                           */
548 /*********************************************/
549
550 int begin_ram ()
551 {
552     /*********************************************************/
553     /*                                                       */
554     /* routine inits amount of free ram                      */
555     /*                                                       */
556     /* heap grows up from static variable storeage area      */
557     /*                                                       */
558     /*********************************************************/
559
560     /* init gobal memchk */
561     memchk=0x07ff;
562     memchk=freeram();
563     startram=memchk;
564
565 } /* end begin_ram */
566
567 /*********************************************/
568 /*                                           */
569 /*********************************************/
570
571 int freeram ()
572 {
573     /*********************************************************/
574     /*                                                       */
575     /* routine returns amount of free ram                    */
576     /*                                                       */
577     /* heap grows up from static variable storeage area      */
578     /* stack grows down from top of memory to the heap       */
579     /* stack ptr == address of last defined auto variable    */
580     /*                                                       */
581     /* __brkval = has value equal to top of the heap         */
582     /* __heap_start = located at bottom of heap ==           */
```

```
583     /*  address of top of statics                          */
584     /*                                                     */
585     /* ATM328 chip has 2K of SRAm = 0x00 to 0x07ff         */
586     /*                                                     */
587     /*******************************************************/
588
589     /* declare external variables */
590     extern int __heap_start, *__brkval;
591
592     /* define autos */
593     int v;
594
595     /* check if top of heap is zero */
596     if(__brkval == 0)
597     {
598           /* compute bottom of stack to top of statics */
599           v = (int) &v - (int) &__heap_start;
600     }
601     else
602     {
603         /* compute bottom of stack to top of heap */
604         v = (int) &v - (int) __brkval;
605     }
606
607     /* save free space if smallest so far */
608     if(v < memchk)
609     {
610         /* debug break here */
611         memchk=v;
612     }
613
614     /* debug */
615     if (v< RAMWRNG)
616     {
617         memchk=v;
618     } /* end if < 5% left */
619
620     /* pass value back */
621     return v;
622 }
623 /* end freeram */
624
625 /*********************************************/
626 /*                                           */
627 /*********************************************/
628
629 void get_stars(int istar,char *st_const, char *st_name,int *rah,int *ram,int *ras,
630                int *decd,int *decm,int *decs, int *retc)
631 {
632     /*********************************************/
633     /*                                           */
634     /* routine returns star details from eprom   */
635     /*                                           */
636     /*********************************************/
637
638     /* define autos */
639
640     /* check search index in range*/
641     *retc = 1;
642     if(istar < 0 || istar >= NSTARS)
643     {
644         *retc = -1;
645         return;
646     } /* end errr return */
647
648     /* write a constellation legend  - copy string text to ram buffer */
649     strcpy_P(st_const, (prog_char *) pgm_read_word(&(cnstell_addr[istar])));
650
651     /* write a star legend  - copy string text to ram buffer */
652     strcpy_P(st_name, (prog_char *) pgm_read_word(&(stname_addr[istar])));
653
654     /* get star coords */
655     coord_ptr = (prog_int16_t *) pgm_read_word(&(stcoord_addr[istar]));
```



```
656     *rah= pgm_read_word(coord_ptr+0);
657     *ram= pgm_read_word(coord_ptr+1);
658     *ras= pgm_read_word(coord_ptr+2);
659
660     *decd= pgm_read_word(coord_ptr+3);
661     *decm= pgm_read_word(coord_ptr+4);
662     *decs= pgm_read_word(coord_ptr+5);
663
664     /* if a -v1 DEC angle, make sure  mins and secs are too */
665     if(*decd<0) *decm = -1*abs(*decm);
666     if(*decm<0) *decs = -1*abs(*decs);
667
668 } /*end get_stars */
669
670 /*********************************************/
671 /*                                           */
672 /*********************************************/
673
674 void gettgt_1(int *endcode)
675 {
676     /*********************************************/
677     /*                                           */
678     /* routine returns chosen target star        */
679     /*                                           */
680     /*********************************************/
681
682     /* define autos */
683     int istar,istar_sav;
684     int rah,ram,ras,decd,decm,decs;
685     int retc,retc2;
686     char st_const[21],st_name[21],st_ax1[21],st_ax2[21];
687
688     /* poll keyboard */
689     retc =1;
690     istar = 0;
691     istar_sav=-1;
692     for (;;)
693     {
694         /* get hold of star data for display */
695         if(istar != istar_sav)
696         {
697             get_stars(istar,st_const, st_name,&rah,&ram,&ras,&decd,&decm,&decs,&retc);
698             istar_sav=istar;
699         }
700
701         /* write scroll/select star data menu */
702         menugo(st_const,st_name,"Scroll^v LFT/RGHT","OK->ENTER Quit->UP",
703             ISKLFT,ISKRGHT,ISKNTR,ISKUP,ISKNULL,&retc);
704
705         /* calibrate backlash */
706         switch(retc)
707         {
708         case 1:
709             /* scroll up */
710             istar--;
711             if(istar<0) istar = NSTARS-1;
712             break;
713         case 2:
714             /* scroll down */
715             istar++;
716             if(istar >= NSTARS) istar = 0;
717             break;
718         case 3:
719             /* select star */
720             sprintf(st_ax1,"RA=%2.1ihr %2.1imn %2.1is",rah,ram,ras);
721             sprintf(st_ax2,"DEC=%2.1idg %2.1imn %2.1is",decd,abs(decm),abs(decs));
722             menugo(st_name,st_ax1,st_ax2,"Align->ENTR Quit->UP",
723             ISKUP,ISKNTR,ISKNULL,ISKNULL,ISKNULL,&retc2);
724
725             istar_sav=-1; /* force a screen update */
726             retc=1;     /*signal continue */
727
728             /* align axes */
```

```
729             if(retc2==2)
730             {
731                 /* signal break */
732                 retc = -1;
733 
734                 /* signal align is not set for duration of measurement */
735                 align_is_set = -1;
736 
737                 /* save index of target star and co-ords*/
738                 align_star_index = istar;
739 
740                 /* save system time of alignement */
741                 align_systime = millis();
742 
743                 /* update - unaligned - current position */
744                 dgrpos(serial1_msk,&rapos,&radir,&ra_dirtxt,
745                 &cor_rapos,&sav_rapos,&cor_radir,
746                 ra_fwdbl,ra_revbl,&ra_degrees, ra_resolve,ra_degtxt,&retc);
747 
748                 dgrpos(serial2_msk,&decpos,&decdir,&dec_dirtxt,
749                 &cor_decpos,&sav_decpos,&cor_decdir,
750                 dec_fwdbl,dec_revbl,&dec_degrees, dec_resolve,dec_degtxt,&retc);
751 
752                 /* compute the dec offset = Ref Star DEC - system DEC */
753                 align_dec_offst = (float) decd + ((float) decm)/60.0 +
754                                 ((float) decs)/3600.0;
755 
756                 /*convert -ve dec to a 360 clock */
757                 if(align_dec_offst < 0.0) align_dec_offst = 360.0 + align_dec_offst;
758 
759                 /* compute offset */
760                 align_dec_offst -= dec_degrees;
761 
762                 /* compute the RA offset - for time now */
763                 align_ra_offst  = (float) rah + ((float) ram)/60.0 +
764                                 ((float) ras)/3600.0;
765                 align_ra_offst  *= 360.0/24.0;
766                 align_ra_offst  -= ra_degrees;
767 
768                 /* signal system is now aligned */
769                 align_is_set = 1;
770 
771                 /* signal exit */
772                 retc =-1;
773             } /* end aligned */
774             break;
775 
776         case 4:
777             /* quit */
778             retc = -1; /* skip out*/
779             break;
780         case 5:
781         default:
782             break;
783         } /* end switch retc */
784 
785         /* see if leave forever loop */
786         if(retc<0) break;
787     } /* end loop forever */
788 
789     /* pick up return code */
790     *endcode = retc;
791 
792 } /* end gettgt */
793 
794 /*********************************************/
795 /*                                           */
796 /*********************************************/
797 
798 void dgrpos(byte chanmsk, long int *chanpos, int *chandir, char **a_chandir,
799             long int *cor_chanpos,long int *savpos,int *cor_chandir,
800             int postoneg, int negtopos,
801             double *axdegr, int resolve, char *degtxt, int *retc)
```



```
802 {
803     /*********************************************/
804     /*                                           */
805     /* routine returns axis position in degrees  */
806     /*                                           */
807     /*********************************************/
808 
809     /* define autos */
810     long unsigned int delta_t;
811     double mins_axis,fdelta_t;
812     float dsign;
813     int iturns;
814 
815     corpos(chanmsk,chanpos,chandir,a_chandir,
816                 cor_chanpos,savpos,cor_chandir,
817                 postoneg,negtopos,retc);
818 
819     /* now convert ticks to degrees */
820     *axdegr  = (double) *cor_chanpos;
821     *axdegr = *axdegr/(double) resolve;
822 
823     iturns = (int) *axdegr;
824     *axdegr = *axdegr - (double) iturns;
825     *axdegr *= 360.0;
826     if(*axdegr < 0) *axdegr = 360.0 + *axdegr;
827 
828     /* if ra axis, divide into hrs/mins/secs */
829     if(chanmsk == ISRAMSK)
830     {
831         /* if axis is aligned -apply any correction */
832         if(align_is_set >0)
833         {
834             *axdegr += align_ra_offst;
835 
836             /* add on any time passed */
837             delta_t =(millis()-align_systime)/1000;
838             fdelta_t =delta_t;
839 
840             /* convert seconds of time to degrees of arc: 1 sec time = 15 secs of arc */
841             fdelta_t *= (15.0/3600.00);
842 
843             /* add on RA time correction - a fixed axis advances 15 sec of RA in 1 sec of time */
844             *axdegr += fdelta_t;
845 
846             /* recondition modulo 360 again */
847             if(*axdegr > 360.0) *axdegr = *axdegr - 360.0;
848             if(*axdegr < 0) *axdegr = 360.0 + *axdegr;
849 
850             /* if DEC in 2nd hemisphere rotate RA 180 degrees */
851             if(dec_is_2ndhs > 0)
852             {
853                 if(*axdegr >= 180.0)
854                     {*axdegr -= 180.0;}
855                 else
856                     { *axdegr += 180.0;}
857             } /* end switch dec hemispheres */
858         } /* end apply RA align offset */
859 
860         mins_axis = *axdegr * (24.0 * 60.0)/360.0;
861         ra_hrs = (int) (mins_axis/60.0);
862         ra_mins = (int) (mins_axis - 60.0 * (float) ra_hrs);
863         ra_secs = (int) (60.0 * (mins_axis - 60.0 * (float) ra_hrs - (float) ra_mins));
864     } /* end if ra axis */
865 
866     /* if dec axis, divide into deg/mins/secs */
867     else
868     {
869         /* make dec axis +ve rotation sense */
870         *axdegr = 360.0 - *axdegr;
871 
872         /* if axis is aligned -apply any correction */
873         if(align_is_set >0)
874         {
```

```
875             *axdegr += align_dec_offst;
876             if(*axdegr < 0.0) *axdegr = 360.0 + *axdegr;
877             if(*axdegr > 360.0) *axdegr -= 360.0;
878 
879             /* aligned dec axis - sort out quadrants */
880             dec_is_2ndhs = -1;
881             if(*axdegr >90.0 && *axdegr <= 270.0)
882             {
883                 *axdegr = 180.0 - *axdegr;
884                 /* set flag for dec in second hemisphere */
885                 dec_is_2ndhs = 1;
886             }
887             else if(*axdegr >270.0 && *axdegr <= 360.0)
888             {
889                 *axdegr = *axdegr - 360.0;
890             }
891         } /* end apply align correction */
892 
893         /* obtain fractional degrees - beware sign */
894         mins_axis = *axdegr * 60;
895         dsign = 1;
896         if(mins_axis < 0.0) dsign = -1.0;
897         mins_axis = abs(mins_axis);
898 
899         /* get mins and secs of absolute value */
900         dec_deg = (int) (mins_axis/60.0);
901         dec_mins = (int) (mins_axis - 60.0 * (float) dec_deg);
902         dec_secs = (int) (60.0 * (mins_axis - 60.0 * (float) dec_deg - (float) dec_mins));
903 
904         /* restore sign and sign components */
905         mins_axis *= dsign;
906         dec_deg *= dsign;
907         dec_mins *= dsign;
908         dec_secs *= dsign;
909     } /* end if dec axis */
910 
911     /* convert float value */
912     dtostrf(*axdegr,DEGLEN,3, degtxt);
913 
914 } /* end dgrpos */
915 
916 /***********************************************/
917 /*                                             */
918 /***********************************************/
919 
920 void mresolve_1(int *endcode)
921 {
922     /***********************************************/
923     /*                                             */
924     /* routine returns runs backlash #1 menu       */
925     /*                                             */
926     /***********************************************/
927 
928     /* define autos */
929     int retc,retc2;
930 
931     /* poll keyboard */
932     for (;;)
933     {
934         /* write resolution menu */
935         menugo("Cal RA tks/rv-> UP","Std RA tks/rv-> DWN","Cal DEC tks/rv-> LFT","Std DEC tks/rv-> RHT" ↙
    ,
936             ISKUP,ISKDWN,ISKLFT,ISKRGHT,ISKNTR,&retc);
937 
938         /* calibrate backlash */
939         switch(retc)
940         {
941         case 1:
942             /* calibrate RA */
943             ra_resolve = RA_RES;
944             break;
945         case 2:
946             /* standard RA */
```



```
947             ra_resolve = RA_RES;
948             sprintf(outstr,"RA tiks/rev=%li",ra_resolve);
949             menugo(outstr,"Values OK-> ENTER","Select again-> UP","\0",
950             ISKUP,ISKNTR,ISKNULL,ISKNULL,ISKNULL,&retc2);
951 
952             retc=1; /*signal continue */
953             if(retc2==2) retc=-1; /*signal break */
954             break;
955         case 3:
956             /* calibrate DEC */
957             dec_resolve = DEC_RES;
958             break;
959         case 4:
960             dec_resolve = DEC_RES;
961             sprintf(outstr,"DEC tiks/rev=%li",dec_resolve);
962             menugo(outstr,"Values OK-> ENTER","Select again-> UP","\0",
963             ISKUP,ISKNTR,ISKNULL,ISKNULL,ISKNULL,&retc2);
964 
965             retc=1; /*signal continue */
966             if(retc2==2) retc=-1; /*signal break */
967             break;
968         case 5:
969             retc = -1; /* skip out*/
970         default:
971             break;
972         } /* end switch retc */
973 
974         /* see if leave forever loop */
975         if(retc<0) break;
976     } /* end loop forever */
977 
978     /* pick up return code */
979     *endcode = retc;
980 
981 } /* end mresolve_1 */
982 
983 /***********************************************/
984 /*                                             */
985 /***********************************************/
986 
987 void mbklash_1(int *endcode)
988 {
989     /***********************************************/
990     /*                                             */
991     /* routine returns runs backlash #1 menu       */
992     /*                                             */
993     /***********************************************/
994 
995     /* define autos */
996     int retc,retc2;
997 
998     /* loop pooling options */
999     for (;;)
1000    {
1001        /* write backlash menu */
1002        menugo("Cal backlash-> UP","Zero b/lash-> DWN","View current-> LFT","Deflt b/lash-> RHT",
1003            ISKUP,ISKDWN,ISKLFT,ISKRGHT,ISKNTR,&retc);
1004 
1005        /* calibrate backlash */
1006        switch(retc)
1007        {
1008        case 1:
1009            /* get RA AC/W backlash */
1010            rabl_acw(serial1_msk,&rapos,&radir,ra_dirtxt,"RA",1,&ra_fwdbl,&retc);
1011            //if(retc<0) break;
1012            rabl_acw(serial1_msk,&rapos,&radir,ra_dirtxt,"RA",-1,&ra_revbl,&retc);
1013            //if(retc<0) break;
1014 
1015            /* get DEC AC/W backlash */
1016            rabl_acw(serial2_msk,&decpos,&decdir,dec_dirtxt,"DEC",1,&dec_fwdbl,&retc);
1017            //if(retc<0) break;
1018            rabl_acw(serial2_msk,&decpos,&decdir,dec_dirtxt,"DEC",-1,&dec_revbl,&retc);
1019            //if(retc<0) break;
```

```
1020
1021            /* signal break */
1022            retc=-1;
1023            break;
1024
1025        case 2:
1026            /* zero backlash parms */
1027            ra_fwdbl =0;
1028            ra_revbl = 0;
1029            dec_fwdbl = 0;
1030            dec_revbl = 0;
1031            sprintf(outstr,"%4i %4i  %4i %4i",ra_fwdbl, ra_revbl, dec_fwdbl, dec_revbl);
1032            menugo("RA Fw-Rev  DC Fw-Rev",outstr,"Values OK-> ENTER","Select again-> UP",
1033            ISKNULL,ISKNULL,ISKNTR,ISKUP,ISKNULL, &retc2);
1034
1035            retc=1; /*signal continue */
1036            if(retc2==3) retc=-1; /*signal break */
1037            break;
1038
1039        case 3:
1040            /* view current parms */
1041            sprintf(outstr,"%4i %4i  %4i %4i",ra_fwdbl, ra_revbl, dec_fwdbl, dec_revbl);
1042            menugo("RA Fw-Rev  DC Fw-Rev",outstr,"Values OK-> ENTER","Select again-> UP",
1043            ISKNULL,ISKNULL,ISKNTR,ISKUP,ISKNULL, &retc2);
1044
1045            retc=1; /*signal continue */
1046            if(retc2==3) retc=-1; /*signal break */
1047            break;
1048
1049        case 4:
1050            /* load default backlash parms */
1051            ra_fwdbl = RA_BL_FWD;
1052            ra_revbl = RA_BL_REV;
1053            dec_fwdbl = DEC_BL_FWD;
1054            dec_revbl = DEC_BL_REV;
1055            sprintf(outstr,"%4i %4i  %4i %4i",ra_fwdbl, ra_revbl, dec_fwdbl, dec_revbl);
1056            menugo("RA Fw-Rev  DC Fw-Rev",outstr,"Values OK-> ENTER","Select again-> UP",
1057            ISKNULL,ISKNULL,ISKNTR,ISKUP,ISKNULL, &retc2);
1058
1059            retc=1; /*signal continue */
1060            if(retc2==3) retc=-1; /*signal break */
1061            break;
1062
1063        case 5:
1064        default:
1065            retc = -1;
1066            break;
1067        } /* end switch retc */
1068
1069        /* see if leave forever loop */
1070        if(retc<0) break;
1071    } /* end loop forever */
1072
1073    /* pick up return code */
1074    *endcode = retc;
1075
1076 } /* end mbklash_1 */
1077
1078 /***********************************************/
1079 /*                                             */
1080 /***********************************************/
1081
1082 void corpos(byte chanmsk, long int *chanpos, int *chandir, char **a_chandir,
1083             long int *cor_chanpos,long int *savpos,int *cor_chandir,
1084             int postoneg, int negtopos,int *retc)
1085 {
1086     /***********************************************/
1087     /*                                             */
1088     /* routine returns position of selected channel */
1089     /*                                             */
1090     /***********************************************/
1091
1092     /* define autos */
```



```
1093
1094     /*get latest chanel tik count and direction */
1095     getpos(chanmsk,chanpos,chandir,a_chandir,retc);
1096
1097     /* get pos change */
1098     *cor_chanpos += (*chanpos-*savpos);
1099     *savpos = *chanpos;
1100
1101     /* see if direction change - correct count */
1102     if(*cor_chandir != *chandir)
1103     {
1104             /* if pos to neg */
1105             if(*cor_chandir >0)
1106             {
1107                 *cor_chanpos -= (long int) postoneg;
1108             } /* end if pos to neg */
1109             else
1110             {
1111                 *cor_chanpos -= (long int) negtopos;
1112             } /* end if neg to pos */
1113
1114             /* save new direction */
1115             *cor_chandir = *chandir;
1116     } /* end if direction change */
1117
1118 } /* end corpos */
1119
1120 /***********************************************/
1121 /*                                             */
1122 /***********************************************/
1123
1124 void menugo(char *s1, char *s2, char *s3, char *s4,
1125 byte kmask1, byte kmask2, byte kmask3,byte kmask4,byte kmask5,int *retc)
1126 {
1127     /***********************************************/
1128     /*                                             */
1129     /* routine runs N option menu and returns      */
1130     /* choice                                      */
1131     /*                                             */
1132     /***********************************************/
1133
1134     /* define autos */
1135     int ipk, test;
1136
1137     /* clear lcd and write msgs */
1138     clrlcd();
1139     lcd_out_msg(0,0,s1);
1140     lcd_out_msg(1,0,s2);
1141     lcd_out_msg(2,0,s3);
1142     lcd_out_msg(3,0,s4);
1143     setrowcol(0,0);
1144     delay(HUMAN);
1145
1146     /* wait for a valid key */
1147     for (;;)
1148     {
1149         /* read keyboard for enter key */
1150         ipk = (int) getkbd();
1151         test = ipk & kmask1;
1152         if(test >0)
1153         {
1154             *retc = 1;
1155             return;
1156         } /* key 1 */
1157
1158         test = ipk & kmask2;
1159         if(test > 0)
1160         {
1161             *retc = 2;
1162             return;
1163         } /* key 2 */
1164
1165         test = ipk & kmask3;
```

```
        if(test > 0)
        {
            *retc = 3;
            return;
        } /* key 3 */

        test = ipk & kmask4;
        if(test > 0)
        {
            *retc = 4;
            return;
        } /* key 4 */

        test = ipk & kmask5;
        if(test > 0)
        {
            *retc = 5;
            return;
        } /* key 5 */
    } /* end wait for a valid key */

} /* end menugo */

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

void rabl_acw(byte imask, long int *axis_pos, int *axis_dir,
              char *dir_txt, char *axis_legend, int fwd, int *backlash, int *ecode)
{
    /***********************************************/
    /*                                             */
    /* routine prompts for axis backlash           */
    /*                                             */
    /***********************************************/

    /* define autos */
    long int axis_sav;
    int retc, blsav,blmean,bln,test;
    char *fwd_txt, *rev_txt;

    /* zero backlash counts */
    bln=0;
    blmean=0;
    *ecode=1;

    /* set desired direction legend */
    fwd_txt = (char *) postxt;
    rev_txt = (char *) negtxt;
    if(fwd < 0)
    {
        fwd_txt = (char *) negtxt;
        rev_txt = (char *) postxt;
    }

    /* loop measuring backlash */
    for (;;)
    {
        /* write messages */
        clrlcd();
        sprintf(outstr,"Go %s %s to target",axis_legend, fwd_txt);
        lcd_out_msg(0,0,outstr);

        lcd_out_msg(2,0,"Ready -> ENTER");
        lcd_out_msg(3,0,"Leave -> UP");

        getpos(imask,axis_pos,axis_dir,&dir_txt, &retc);
        sprintf(outstr,"%s= %-10.1li %s",axis_legend,*axis_pos,dir_txt);
        lcd_out_msg(1,0,outstr);

        /* pause */
        delay(HUMAN);

```



```
        /* loop printing axis value */
        for (;;)
        {
            getpos(imask,axis_pos,axis_dir,&dir_txt,&retc);
            sprintf(outstr,"%s= %-10.1li %s",axis_legend,*axis_pos,dir_txt);
            lcd_out_msg(1,0,outstr);

            /* read keyboard for enter key */
            retc = (int) getkbd();
            test = retc & ISKNTR;
            if(test > 0)
            {
                /* make sure movement in right direction direction */
                if(*axis_dir == fwd) break;

                /* write err msg */
                lcd_out_msg(2,0,"Err-wrong direction ");
                delay(HUMAN);

                /* continue */
                lcd_out_msg(2,0,"Ready -> ENTER      ");
            } /* end test valid enter */

            /* test leave */
            test = retc & ISKUP;
            if(test > 0)
            {
                *ecode=-1;
                return;
            }
        } /* end wait for enter/leave */

        /* reposition target from other side */
        axis_sav=*axis_pos;
        clrlcd();

        /* write messages */
        sprintf(outstr,"%s %s= %-10.1li",fwd_txt,axis_legend,axis_sav);
        lcd_out_msg(0,0,outstr);

        sprintf(outstr,"Go %s to target",rev_txt);
        lcd_out_msg(1,0,outstr);

        lcd_out_msg(3,0,"Ready-ENTER Leave-UP");

        sprintf(outstr,"%s= %-10.1li %s",axis_legend,*axis_pos,dir_txt);
        lcd_out_msg(2,0,outstr);

        /* pause */
        delay(HUMAN);

        /* loop printing axis value */
        for (;;)
        {
            getpos(imask,axis_pos,axis_dir,&dir_txt,&retc);
            sprintf(outstr,"%s= %-10.1li %s",axis_legend,*axis_pos, dir_txt);
            lcd_out_msg(2,0,outstr);

            /* read keyboard for enter key */
            retc = (int) getkbd();
            test = retc & ISKNTR;
            if(test > 0)
            {
                /* make sure move in reverse direction */
                if(*axis_dir != fwd) break;

                /* write err msg */
                lcd_out_msg(3,0,"Err-wrong direction ");
                delay(HUMAN);

                /* continue */
                lcd_out_msg(3,0,"Ready-ENTER Leave-UP");
            } /* end test valid enter */
```

```
1312
1313            /* test leave */
1314            test = retc & ISKUP;
1315            if(test > 0)
1316            {
1317                *ecode=-1;
1318                return;
1319            }
1320        } /* end wait for enter */
1321
1322        /* get backlash value and new mean value */
1323        blsav= (int) (*axis_pos-axis_sav);
1324        blmean = blmean * bln + blsav;
1325        bln++;
1326        blmean=blmean/bln;
1327        *backlash=blmean;
1328
1329        /* ask again */
1330        clrlcd();
1331
1332        /* write messages */
1333        sprintf(outstr,"%s B/L %s = %i",fwd_txt,axis_legend,blsav);
1334        lcd_out_msg(0,0,outstr);
1335
1336        sprintf(outstr,"Mean B/L = %i",blmean);
1337        lcd_out_msg(1,0,outstr);
1338        lcd_out_msg(2,0,"Exit -> ENTER");
1339        lcd_out_msg(3,0,"Repeat -> UP");
1340
1341        /* pause */
1342        delay(HUMAN);
1343
1344        /* read keyboard for enter key */
1345        for (;;)
1346        {
1347            retc = (int) getkbd();
1348            retc &= ISKUP;
1349            if(retc > 0) break;
1350
1351            retc = (int) getkbd();
1352            retc &= ISKNTR;
1353            if(retc > 0) break;
1354        }
1355
1356        /* if repeat */
1357        retc &= ISKUP;
1358        if (retc >0) continue;
1359
1360        /* must be enter key - leave */
1361        break;
1362
1363    } /* end loop measuring backlash */
1364
1365 } /* end rabl_acw */
1366
1367 /*********************************************/
1368 /*                                           */
1369 /*********************************************/
1370
1371 byte getkbd()
1372 {
1373     /*********************************************/
1374     /*                                           */
1375     /* routine debounces keyboard switches       */
1376     /*                                           */
1377     /*********************************************/
1378
1379     /* define autos */
1380     byte key1,key2;
1381
1382     /* loop until same answer twice */
1383     for (;;)
1384     {
```



```
1385            key1=getkbd2();
1386            delay(DEBOUNCE);
1387            key2=getkbd2();
1388            if(key1 == key2) break;
1389    } /* end read same answer twice */
1390
1391 } /* end getkbd */
1392
1393 /*********************************************/
1394 /*                                           */
1395 /*********************************************/
1396
1397 byte getkbd2()
1398 {
1399     /*********************************************/
1400     /*                                           */
1401     /* routine reads keyboard switches           */
1402     /*                                           */
1403     /*********************************************/
1404
1405     /* define autos */
1406     byte bitin, keypad;
1407
1408     /* read up key */
1409     keypad = 0;
1410     bitin = PINC & key_up_msk;
1411     if(bitin < 1) keypad += ISKUP;
1412
1413     /* down */
1414     bitin = PIND & key_dwn_msk;
1415     if(bitin < 1) keypad += ISKDWN;
1416
1417     /* left */
1418     bitin = PIND & key_lft_msk;
1419     if(bitin < 1) keypad += ISKLFT;
1420
1421     /* right */
1422     bitin = PINC & key_rght_msk;
1423     if(bitin < 1) keypad += ISKRGHT;
1424
1425     /* enter */
1426     bitin = PINC & key_ntr_msk;
1427     if(bitin < 1) keypad += ISKNTR;
1428
1429     return(keypad);
1430
1431 } /* end getkbd2 */
1432
1433 /*********************************************/
1434 /*                                           */
1435 /*********************************************/
1436
1437 void getpos(byte chanmsk, long int *chanpos, int *chandir, char **a_chandir, int *retc)
1438 {
1439     /*********************************************/
1440     /*                                           */
1441     /* routine returns position of selected channel */
1442     /*                                           */
1443     /*********************************************/
1444
1445     /* define autos */
1446     long int poscnt;
1447     int i;
1448     byte tchar[11], *tptr;
1449
1450     /* poll serial */
1451     for (i=0; ;)
1452     {
1453         /* poll serial in */
1454         tptr=tchar+i;
1455         suckc(chanmsk,tptr,retc);
1456         if (*retc > 0)
1457         {
```

```
            /* valid char - test conversion if eof */
            if(*tptr == ',')
            {
                /* convert position string to a long int */
                makbin((char *) tchar,i+1,&poscnt,retc);

                /* return binary count if an OK conversion */
                if(*retc>0)
                {
                    /* update direction - compare with last version */
                    if(poscnt < *chanpos) *chandir = -1;

                    if(poscnt > *chanpos) *chandir = 1;

                    /* save return value */
                    *chanpos = poscnt;
                    *a_chandir = (char *) postxt;
                    if(*chandir < 0) *a_chandir = (char *) negtxt;

                    return;
                } /* end valid conversion */

                /* start assembling next string if error conversion */
                i=0;
                continue;
            } /* end if new sequence */

            /* inc char count */
            i++;
            if(i>10) i=0; /* safety catch if formatting lost */

            /* write char to lcd */
            /*wbytdr(*tptr); */
        } /* end if a char in */

    } /* end poll serial forever */

} /* end getpos */

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

void makbin(char *instr, int len, long int *outi, int *retc)

{
    /*********************************************/
    /*                                           */
    /* routine converts an encoder input string to  */
    /* long int position count.                  */
    /*                                           */
    /*********************************************/

    /* define autos */
    int i, xint;
    char  xchar;

    /* test if string valid format "0xhhhhhhhh," */
    *retc = 1;
    *outi = 0;
    if (len < 11)
    {
            *retc = -1;
            return;
    } /* end string too short */

    /* test string begins '0x' */
    if(*instr == '0' && *(instr+1) == 'x')
    {
        /* now stuff digits into long integer */
        for (i = 2; i < 10; i++)
        {
            xchar=*(instr+i);
```



```
            xint = (int) xchar - 0x30;

            /* convert a to f */
            if(xint > 9) xint -= 0x27;

            /* multiply u the digit */
            *outi = *outi * (long int) 0x10 + (long int) xint;

        } /* end swap chars to digits */
    } /* end extract digits */
    else
    {
            *retc = -2;
            return;
    } /* missing precursor "0x" */

} /* end makbin */

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

void suckc(byte smask, byte *msg, int *retc)
{
    /*********************************************/
    /*                                           */
    /* routine reads a byte from serial port     */
    /*                                           */
    /*********************************************/

    /* define autos */
    unsigned long int mus, mus2, dift;
    int i;
    byte bitin;

    /* check memory */
    freeram();

    /* loop waiting for a low - hopefully a start bit */
    mus=micros();
    *retc = 1;
    *msg = 0x0;
    for (;;)
    {
        /* get elapsed time */
        mus2 = micros();
        if(mus2 < mus) mus = mus2; /* check if clock rolled over */

        /* test if polling time expired */
        dift = mus2 - mus;
        if(dift > (unsigned long int) MAX_SER_WAIT)
        {
                *retc=-1;
                return;
        } /* end if wait time expired and no data */

        /* test input pin */
        bitin = PINB & smask;

        /* if start bit found - leave loop */
        if(bitin < 1) break;

    } /* end polling loop */

    /* got a start bit - wait till middle of it */
    delayMicroseconds(DLA9600_2);

    /* now loop reading in bits - we hope */
    for (i = 0; i< 8; i++)
    {
        /* signal debug */
        PORTB &= clrsig; /* put line lo */
```

```
1604         /* wait a bit time */
1605         delayMicroseconds(DLA9600_2);
1606
1607         /* signal debug */
1608         PORTB |= setsig; /* put line hi */
1609
1610         /* wait a bit time */
1611         delayMicroseconds(DLA9600_2);
1612
1613         /* get bit, move into position and save it */
1614         bitin = 0;
1615         if(PINB & smask) bitin=1;
1616         bitin = bitin << i;
1617         *msg |= bitin;
1618     } /* end loop over 8 bit times */
1619
1620     /* wait out 1 stop bits and finish */
1621     delayMicroseconds(DLA9600);
1622
1623 } /* end suckc */
1624
1625 /*********************************************/
1626 /*                                           */
1627 /*********************************************/
1628
1629 void lcd_out_msg (int row, int col,char *msg)
1630 {
1631     /*********************************************/
1632     /*                                           */
1633     /* routine lcd_out_msg - write a string to lcd  */
1634     /*                                           */
1635     /*********************************************/
1636
1637     /* define autos */
1638     char *mptr;
1639     int i, mlen;
1640
1641     /* position lcd write pt */
1642     setrowcol(row,col);
1643
1644     /* check length of string */
1645     mlen=strlen(msg);
1646
1647     /* loop writing chars */
1648     if(mlen >0)
1649     {
1650         /* print string */
1651         for (mptr=msg, i=0; i< mlen; mptr++, i++)
1652         {
1653                 wbytdr(byte(*mptr));
1654         } /* end loop over msg len */
1655
1656         /*now blank rest of line */
1657         //for (i=mlen; i< 20; i++)
1658         //{
1659                 //wbytdr(byte(0x20));
1660         //} /* end loop over msg len to end of line */
1661     }
1662     else
1663     {
1664         /* null string -=> write a blank line */
1665         for (i=0; i< 20; i++)
1666         {
1667                 wbytdr(byte(0x20));
1668         } /* end loop over msg len */
1669     }
1670
1671 } /* end lcd_out_msg */
1672
1673 /*********************************************/
1674 /*                                           */
1675 /*********************************************/
1676
```



```
1677 void setrowcol(int row,int col)
1678 {
1679     /*********************************************/
1680     /*                                           */
1681     /* routine positions DDRAM address at row (0-3) */
1682     /* and col (0-19)                            */
1683     /*                                           */
1684     /*********************************************/
1685
1686     /* define autos */
1687     byte pos;
1688
1689     /* convert row,col to barmy lcd address */
1690     switch (row)
1691     {
1692     case 0:
1693         pos=0x0;
1694         break;
1695     case 1:
1696         pos = 0x40;
1697         break;
1698     case 2:
1699         pos = 0x14;
1700         break;
1701     case 3:
1702         pos = 0x54;
1703         break;
1704     default:
1705         pos = 0x0;
1706         break;
1707     } /* end switch row */
1708
1709     /* convert column */
1710     if (col >19)
1711     {
1712         col = 19;
1713     } /* end check column */
1714     if (col <0)
1715     {
1716         col = 0;
1717     } /* end check column */
1718
1719     /* add on the column number */
1720     pos += (byte) col;
1721
1722     /* Instr reg byte: set DDRAM address 0x80 + address */
1723     pos += 0x80;
1724     wbytir(pos);
1725
1726 } /* end setrowcol */
1727
1728 /*********************************************/
1729 /*                                           */
1730 /*********************************************/
1731
1732 void mak_lcd_lut(byte *lcd_lut_ptr)
1733 {
1734     /*********************************************/
1735     /*                                           */
1736     /*  routine creates look up table to convert an */
1737     /*  i/p nibble to an output nibble on LCD data  */
1738     /*  lines                                    */
1739     /*                                           */
1740     /*********************************************/
1741
1742     /* define autos */
1743     int i;
1744     byte j, mask,sum;
1745
1746     for (i = 0; i<16; i++)
1747     {
1748         j = i;
1749
```

```
        /* test each of lowest 4 bits */
        sum=0;
        mask = 0x01;
        if(j & mask)
        {
            sum +=lcd_d4_msk;
        } /* end set ls bit */

        mask <<=1;
        if(j & mask)
        {
            sum +=lcd_d5_msk;
        } /* end set d5 bit */

        mask <<=1;
        if(j & mask)
        {
            sum +=lcd_d6_msk;
        } /* end set d6 bit */

        mask <<=1;
        if(j & mask)
        {
            sum +=lcd_d7_msk;
        } /* end set d7 bit */

        /* save translated code */
        *(lcd_lut_ptr +i)=sum;

    } /* end loop over lookup tables */

    /* now make nibble to clear down lcd data bus - PORT & lcd_clear = cleared bits */
    lcd_xclear = lcd_d4_msk+lcd_d5_msk+lcd_d6_msk + lcd_d7_msk;
    lcd_xclear = ~lcd_xclear;

} /* end mak_lcd_lut */

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

void init_lcd()
{

    /***********************************************/
    /*                                             */
    /* routine initialises lcd module for 4 bit ops */
    /*                                             */
    /***********************************************/

    /* define autos */
    byte tchar;

    /* wait > 40 msec after start up */
    delay(50);

    /* kick off 1 nibble to Instruction Reg: fn set cmd, 8 bits */
    tchar=0x02+0x01;
    wnibir(tchar);

    /* wait > 39 us */
    delayMicroseconds(50);

    /* Instr reg byte: fn set cmd, 4bit bus, 4 lines, 5x8 font */
    tchar=0x20+0x08;
    wbytir(tchar);

    /* do it again */
    tchar=0x20+0x08;
    wbytir(tchar);

    /* Instr reg byte: display set: Display ON, Cursor ON, Blink ON */
    tchar=0x08 + 0x04 + 0x02 + 0x01;
```



```
    wbytir(tchar);

    /* Instr reg byte: clear display */
    tchar=0x01;
    wbytir(tchar);
    delay(2); /* wait more than 1.53 msec */

    /* Instr reg byte: set entry mode: cursor direction, shift display */
    tchar = 0x04;
    wbytir(tchar);

    /* clear Data ram and return cursor to home */
    tchar=0x02;
    wbytir(tchar);
    delay(2); /* wait more than 1.53 msec */

} /* end init lcd */

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

void clrlcd()
{

    /***********************************************/
    /*                                             */
    /* routine Clears Display - DDRAM and DDRAM add */
    /*                                             */
    /***********************************************/

    /* define autos */
    /* Instr reg byte: clear display */
    wbytir(0x01);
    delay(2); /* wait more than 1.53 msec */

} /* end clear display */


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

void wbytir(byte s)
{

    /***********************************************/
    /*                                             */
    /* routine writes a byte to lcd instruction    */
    /* register IR                                 */
    /*                                             */
    /***********************************************/

    /* define autos */
    byte hinib, lonib;

    /* get high order nibble */
    hinib = s >> 4;
    lonib = s & 0x0f;
    wnibir(hinib);
    wnibir(lonib);

    /* default 50 us */
    delayMicroseconds(50);

} /* end wbytir */

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

void wnibir(byte s)
```

```
{

    /*********************************************/
    /*                                           */
    /* routine writes a nibble to lcd instruction   */
    /* register IR                               */
    /*                                           */
    /*********************************************/

    /* define autos */
    byte hinib, lonib, xlate;

    /* select instruction register */
    select_ireg();

    /* select write mode */
    select_write();

    /* translate the nibble */
    xlate = *(lcd_lut+(int) s);

    /* write to Port D - lcd data bits */
    PORTD &= lcd_xclear; /* clear data bus bits */
    PORTD |= xlate;  /* set any desired lcd data bits */

    /* now cycle the lcd enable latch */
    cycle_ena_latch();

} /* end wnibir */

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

void wbytdr(byte s)
{

    /*********************************************/
    /*                                           */
    /* routine writes a byte to lcd data            */
    /* register DR                               */
    /*                                           */
    /*********************************************/

    /* define autos */
    byte hinib, lonib;

    /* test memory */
    freeram();

    /* get high order nibble */
    hinib = s >> 4;
    lonib = s & 0x0f;
    wnibdr(hinib);
    wnibdr(lonib);

    /* default 50 us */
    delayMicroseconds(50);

} /* end wbytdr */

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

void wnibdr(byte s)
{

    /*********************************************/
    /*                                           */
    /* routine writes a nibble to lcd data          */
    /* register IR                               */
    /*                                           */
```



```
    /*********************************************/

    /* define autos */
    byte hinib, lonib, xlate;

    /* select instruction register */
    select_dreg();

    /* select write mode */
    select_write();

    /* translate the nibble */
    xlate = *(lcd_lut+(int) s);

    /* write to Port D - lcd data bits */
    PORTD &= lcd_xclear; /* clear data bus bits */
    PORTD |= xlate;  /* set any desired lcd data bits */

    /* now cycle the lcd enable latch */
    cycle_ena_latch();

} /* end wnibdr */

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

void cycle_ena_latch()
{

    /*********************************************/
    /*                                           */
    /* routine cycles the LCD Enable latch from     */
    /* low to high to low.                       */
    /*                                           */
    /*********************************************/

    /* assume PORT B - put mask bit low */
    PORTB &= ~(lcd_ena_msk);

    /* put mask bit high */
    PORTB |= lcd_ena_msk; /* high pulse for 40 nsec */

    /* put mask bit low */
    PORTB &= ~(lcd_ena_msk); /* data still valid for 10 nsec */

} /* end cycle_ena_latch */

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

void select_ireg()
{

    /*********************************************/
    /*                                           */
    /* routine selects lcd Instruction Register     */
    /* Register Select line pulled low           */
    /*                                           */
    /*********************************************/

    /* assume PORT B - put mask bit low */
    PORTB &= ~(lcd_rs_msk);

} /* end select_ireg */

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

void select_dreg()
{
```

```
2042
2043     /***********************************************/
2044     /*                                             */
2045     /* routine selects lcd Data Register           */
2046     /* Register Select line pulled high            */
2047     /*                                             */
2048     /***********************************************/
2049
2050     /* assume PORT B - put mask bit high */
2051     PORTB |= lcd_rs_msk;
2052
2053 } /* end select_dreg */
2054
2055 /***********************************************/
2056 /*                                             */
2057 /***********************************************/
2058
2059 void select_write()
2060 {
2061
2062     /***********************************************/
2063     /*                                             */
2064     /* routine selects write mode to LCD           */
2065     /* R/W line pulled low                         */
2066     /*                                             */
2067     /***********************************************/
2068
2069     /* assume PORT B - put mask bit low */
2070     PORTB &= ~(lcd_rw_msk);
2071
2072 } /* end select_write */
2073
2074 /***********************************************/
2075 /*                                             */
2076 /***********************************************/
2077
2078 void select_read()
2079 {
2080
2081     /***********************************************/
2082     /*                                             */
2083     /* routine selects read mode from LCD          */
2084     /* R/W line pulled high                        */
2085     /*                                             */
2086     /***********************************************/
2087
2088     /* assume PORT B - put mask bit high */
2089     PORTB |= lcd_rw_msk;
2090
2091 } /* end select_read */
2092
2093 /***********************************************/
2094 /*                                             */
2095 /***********************************************/
2096
```