```
/* examine.c - version 1.3 */

/*****************************************************************************
 *
 *
 * PROGRAM:        examine.c
 *
 * Programmer:     Garry Daly
 *
 * Company:        TBD
 *
 * Created:        Mon Apr 26 07:45:00
 *
 * Abstract:       Analyze the Software application output
 *
 * Description:    This program will analyze the results of the software
 *                 application execution.
 *
 * Contents:
 *
 * Modified:
 *
 *   DATE    DEV                          CHANGE ABSTRACT
 * ........  ...   ..........................................................
 * 04/26/xx  grd   ver. 0.0   - program was born.
 * 09/15/xx  grd   ver. 0.1   - Bug fix version.  Selection algolrithm
 *                            - modified
 * 12/21/xx  grd   ver. 0.2   - Corrected heap sort bug and added date/time
 *                            - Add unique error message listings
 *                            - Add package comparison functionality
 * 02/09/xx  grd   ver. 0.3   -
 * 04/21/xx  grd   ver. 0.4.1 - Added flexibility to add a caption and center it
 * 05/02/xx  grd   ver. 0.4.2 - Added flexibility to add a generic
 * 08/16/xx  grd   ver. 0.4.3 - Added Stabilility View Analysis
 * 09/08/xx  grd   ver. 0.4.4 - Added Test Id Number to the Error Report
 * 10/14/xx  grd   ver. 0.5   - Modified error message handling to create an
 *                            - error dictionary and store indices for
 *                            - comparisons. All the errors reported from each
```

```
 *                            - operation together are considered an error
 *                            - cluster where in the past, only the first error
 *                            - of the cluster was reported.
 * 09/26/xx  grd   ver. 0.5.1 - Support was added for software versions and the
 *                            - history file is read for start/stop run times.
 * 06/28/xx  grd   ver. 1.1   - Placed under sccs and modified for ansi C
 *
 * 03/14/xx  grd   Modified to accept 5 character official package names
 *                 GOAT ITEM 1726.
 # 04/04/xx  grd   Modified to allow upper case 3000 names
 #                 Goat item 1753
 *
 ************************************************************************/

char Version[] = "1.3";
char Delta[]   = "4/5/97 - 08:56:40";

#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>

/*
 *
 * FILE:        TSTexamine.h
 *
 * Programmer:  G. R. Daly
 *
 * Company:     TBD
 *
 *
 * DESCRIPTION:          This header file contains the define constants and externs
 *                       for the examine() software which examines application output.
 *
 *
 * SUPPORTING DOCUMENTS: See the Application User Guide for more information
 *                       regarding application output.
```

```
 *
 */

#ifndef _TSTEXAMINE


          /************************************/
          /* GLOBAL EXTERNAL PROGRAM VARIABLES */
          /************************************/

extern   int       errno;          /* UNIX global error number           */


              /*********************/
              /* LOCAL DEFINITIONS */
              /*********************/

#define OUTSUFFIX  ".out"
#define ERRSUFFIX  ".err"
#define RPTSUFFIX  ".rpt"
#define RCSUFFIX   ".RC"
#define HDRSTR0    "For"
#define HDRSTR1    "SU  Package:"
#define HDRSTR2    "vs. Package:"
#define HDRSTR3    "RC:"
#define MINUS      "-"
#define DOTRCDOT   ".RC."
#define DOTHISTDOT ".HIST."
#define SPPIPESP   " | "
#define LASTEDIR   "last_perr"
#define OLDEDIR    "old_perr"
#define FORM       " form="
#define LBFORM     "# form="
#define NEW        "new!"
#define CHG        "chg!"
#define OUT        "out!"
#define VFY        "vfy!"
#define MVFY       "mvfy!"
#define SHVFY      "shvfy!"
```

```
118  #define ATTR       "attr!"
119  #define UNKWN      "unkwn!"
120  #define STOP       " #stop"
121  #define SCREEN     " #screen "
122  #define EXIT       " #exit"
123  #define PF         ":  pf"
124  #define SPASS      "PASS"
125  #define SFAIL      "FAIL"
126  #define SNOCARE    "NOCARE"
127  #define MSGU       "MSG!"
128  #define MSGL       "msg!"
129  #define SCRIPTNM   "Application: using input file"
130  #define TIDPFX     "uc"
131  #define STARTTID   "starttid"
132  #define EXPASS     "Application: a pass was expected"
133  #define EXFAIL     "Application: a fail was expected"
134  #define MEXFAIL    "Application: a FAIL was expected"
135  #define MNOMORE    "?E no more error messages"
136  #define SCREEN1FMT "\n****************** ERROR:  EXPECTED TEST RESULT:  %-10s ******************\n"
137  #define DASHES     "-------------------------------------------------------------------------------
138  "
139  #define BLANKS     "          \n"
140
141  #define FALSE      0
142  #define TRUE       1
143
144  #define CAPLIMIT   26
145  #define GENLIMIT   6
146  #define MAXPFX     12
147  #define RPTWIDTH   80
148
149  #define PRTMAX1    106
150
151  #define NOSTRFOUND 0
152  #define MAXLINENO  300
153  #define MAXREAD    256
154
155  #define TOTCLASS   30
156  #define TOTVIEW    70
```

```
157
158  #define MAXCLASS   TOTCLASS + 3
159  #define MAXVIEW    TOTVIEW  + 1
160
161  #define CLASSSZ    5
162  #define VIEWSZ     5
163  #define OPERSZ     10
164
165  #define MAXSCRERR  1000
166  #define NAMESZ     20
167  #define TIDSZ      12
168
169  #define MAXERRMSG  1001
170  #define ERRMSGSZ   81
171
172  #define DAYSEC     86400
173  #define HRSEC      3600
174  #define MINSEC     60
175
176
177               /***************************/
178               /* LOCAL DATA DECLARATIONS */
179               /***************************/
180
181  FILE *Ifp;                        /* input file one pointer     */
182  FILE *Hfp;                        /* history file pointer       */
183  FILE *Vfp;                        /* error view number file ptr */
184  FILE *Sfp;                        /* error script name file ptr */
185  FILE *Efp;                        /* unique errors file pointer */
186  FILE *Rfp;                        /* report file pointer        */
187  FILE *Ofp;                        /* output run file pointer    */
188
189  char  Inbuf[MAXLINENO][MAXREAD];       /* input buffer              */
190  char  Cur_view[VIEWSZ];                /* (char)  View  number      */
191  char  Cur_class[CLASSSZ];              /* (char)  Class number      */
192
193  short Cur_viewnum;                /* (short) View  number       */
194  short Cur_classnum;               /* (short) Class number       */
195  short Vlog;                       /* log errors by view number  */
```

```
196  short Slog;                        /* log errors by script name  */
197  short Olog;                        /* log errors in output file  */
198  short Errflag;                     /* denotes if pkg has errors  */
199
200  int   Serr_cnt;                         /* count of scripts in error  */
201  long  Linesread;                   /* count of the RC lines read */
202  long  Tlines;                      /* count of the RC lines read */
203  long  Tstfail;                     /* count of RC failures       */
204  long  Tstpass;                     /* count of RC successes      */
205
206  long  Pnew;                        /* count of successful new    */
207  long  Pchg;                        /* count of successful chg    */
208  long  Pout;                        /* count of successful out    */
209  long  Pvfy;                        /* count of successful vfy    */
210  long  Pmvfy;                       /* count of successful mvfy   */
211  long  Pshvfy;                      /* count of successful shvfy  */
212  long  Pattr;                       /* count of successful attr   */
213  long  Punkwn;                      /* count of successful unknwn */
214
215  long  Fnew;                        /* count of failed     new    */
216  long  Fchg;                        /* count of failed     chg    */
217  long  Fout;                        /* count of failed     out    */
218  long  Fvfy;                        /* count of failed     vfy    */
219  long  Fmvfy;                       /* count of failed     mvfy   */
220  long  Fshvfy;                      /* count of failed     shvfy  */
221  long  Fattr;                       /* count of failed     attr   */
222  long  Funkwn;                      /* count of failed     unknown*/
223
224  long  Tottests;                         /* count of RC attempts       */
225
226               /*************************/
227               /* LOCAL DATA STRUCTURES */
228               /*************************/
229
230  struct rcscripts                   /* rc scripts in error        */
231  {
232      char  name[NAMESZ];
233      int   count;
234
```

```
} Rc[MAXSCRERR];

struct dictionary                   /* rc error messages          */
{
    struct odinmsg
    {
        char  em1[ERRMSGSZ];            /* error message line 1       */
        char  em2[ERRMSGSZ];            /* error message line 2       */

    } msg[MAXERRMSG + 1];

    int   entries;                  /* count of dictionary entries*/

} Errdict;

struct curpkg                       /* new package error messages */
{
    struct cur_err
    {
        char  errindx[ERRMSGSZ];        /* actual indexes into dict   */
        int   no_entries;               /* number of times error found*/
        int   no_indexes;               /* number of error indexes    */
        int   no_lines;                 /* number of printable lines  */

    } nerr[MAXERRMSG + 1];

    int   entries;                  /* count of error numbers     */

} Nperr;

struct oldpkg                       /* Old package error messages */
{
    struct old_err
    {
        char  errindx[ERRMSGSZ];
        int   no_indexes;               /* number of error indexes    */
        int   no_lines;                 /* number of printable lines  */

    } oerr[MAXERRMSG + 1];
```

```
274
275     int   entries;                /* count of error numbers     */
276
277 } Operr;
278
279 struct rcclass
280 {
281     struct rcviews
282     {
283         struct tstpass
284         {
285             int new;
286             int chg;
287             int out;
288             int vfy;
289             int mvfy;
290             int shvfy;
291             int attr;
292             int unkwn;
293             int count;
294         } Pass;
295
296         struct tstfail
297         {
298             int new;
299             int chg;
300             int out;
301             int vfy;
302             int mvfy;
303             int shvfy;
304             int attr;
305             int unkwn;
306             int count;
307         } Fail;
308
309     } View[MAXVIEW];
310
311 } Class[MAXCLASS];
312
```

```
struct rchist
{
    char  filename[80];              /* input history file name    */

    char  start[80];                 /* stream history start       */
    char  end[80];                /* stream history end         */

    char  start_day[10];          /* stream history start day   */
    int   smon;                      /* start month of run         */
    int   sday;                      /* start day   of run         */
    int   syr;                    /* start year  of run         */

    char  start_time[10];            /* stream history start time  */
    int   shr;                    /* start hour  of run         */
    int   smin;                      /* start min   of run         */
    int   ssec;                      /* start sec   of run         */

    char  end_day[10];               /* stream history start day   */
    int   emon;                      /* end   month of run         */
    int   eday;                      /* end   day   of run         */
    int   eyr;                    /* end   year  of run         */

    char  end_time[10];              /* stream history start time  */
    int   ehr;                    /* end   hour  of run         */
    int   emin;                      /* end   min   of run         */
    int   esec;                      /* end   sec   of run         */

    int   days;                      /* run duration in days       */
    int   hours;                  /* run duration in hours      */
    int   mins;                      /* run duration in minutes    */
    int   secs;                      /* run duration in seconds    */

    long  start_sec;                 /* starting  time in seconds  */
    long  end_sec;                /* ending    time in seconds  */
    long  run_sec;                /* run       time in seconds  */
    long  remain;                 /* remainder time in seconds  */

} Runhist;
```

```
352   char  Out_file[80];           /* output file name           */
353   char  Rpt_file[80];           /* report file name           */
354   char  Err_file[80];           /* unique error file name     */
355
356   #define _TSTEXAMINE
357   #endif
358
359
360
```

```
/*****************************************************************************
 *
 * ************
 * *** FILE ***            PROLOGUE:
 * ************
 *
 * File:                   TSTexamine.c
 *
 * Description:            This examines the application output for
 *                         for failures of expected pass and expected fail
 *                         conditions.
 *
 * Supporting Documents:   None.
 *
 *
 * Contents:               TSTmain()     - Opens the Application output file as input
 *                                         to this program and analyzes it.
 *                         sindex()      - Searches for one character string
 *                                         within another.
 *                         prcs_err()    - Proceses error messages.
 *                         crheap()      - Creates a heap array for heap sort.
 *                         bldindx()     - Builds an string of error indexes
 *                         inserr()      - Inserts error msgs into dictionary
 *
 *****************************************************************************/
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:               TSTmain()
 *
 * Abstract:           This function open the application output file as its input.
 *                     It will look for Applications that failed when
 *                     expected to pass and Applications that pass when
 *                     expected to fail.
 *
 *    Programmer(s):   G. R. Daly
 *
 *          Company:   TBD
 *             Date:   Mon Apr 26 07:45:00 CST
 *
 *
 * Loadable package:   None.
 *
 * Usage:              examine rc_su_package
 *
 *
 * Parameters:         None.
 *
 * Externals:          None.
 *
 *
 * Returns:            Return Value(s) -
 *
 *                     FAIL (1)   - exit with return code 1 and displays
 *                                  an error message.
 *
 *
 * Description:        This function examines an application output file.

 *
 * Implementation:     Standard UNIX.
```

```
428    *
429    *
430    * Warnings:           None.
431    *
432    * Examples:           None.
433    *
434    * Calls:              sprintf()   - Formats characters strings
435    *                     strrchr()   - Finds last string occurance
436    *                     strcpy()    - copies one string to another
437    *                     strncpy()   - copies n characters from one string
438    *                     strcmp()    - compares one character string to another
439    *                     strcat()    - concatenates one string to another
440    *                     fgets()     - system read call for 1 line
441    *                     tolower()   - system lower case conversion function
442    *                     getopt()    - gets command line options
443    *                     fclose()    - closes stream files
444    *                     setbuf()    - set buffer for streams
445    *                     fprintf()   - Formats stream output
446    *                     sprintf()   - Formats character strings
447    *                     exit()      - exits program
448    *                     fopen()     - opens a file for stream
449    *                     sindex()    - looks for character strings
450    *                     prcs_err()  - writes rc errors to a file
451    *
452    * Macros:             None.
453    *
454    *
455    * See Also:           Application Usage Guide.
456    *
457    ************************************************************************/
458
459
```

```
/* BOF - TSTmain */

int
main(argc, argv)
int   argc;
char *argv[];
{
     /* System Calls */

     extern int   getopt();             /* gets command line options  */
     extern int   fclose();             /* closes stream files        */
     extern int   strcmp();             /* compares character strings */
     extern long  time();               /* Unix system time call      */
     extern char *ctime();              /* current system date/time   */
     extern char *strcpy();             /* copies character strings   */
     extern char *strncpy();            /* copies character strings   */
     extern char *strrchr();            /* finds last string occurance*/
     extern char *strcat();             /* concatenates strings       */
     extern char *fgets();              /* gets stream lines of input */
     extern FILE *fopen();              /* opens a file for stream    */
     extern DIR  *opendir();            /* opens directories          */

     /* Local Calls */

     int   sindex();                    /* looks for character strings*/
     void  crheap();                    /* prepares date for heap sort*/
     void  prcs_err();             /* writes rc errors to a file */
     void  updstat();              /* updates stats for tests    */
     void  capcntr();              /* centers captions passed    */
     void  bldindx();              /* builds error msg indexes   */

     /* Global Variables */

     extern int   optind;               /* used with getopt()         */
     extern char *optarg;               /* used with getopt()         */

     /* Local Variables */

     char  lastscreen[MAXREAD];         /* current screen command     */
```

```
499         char  elastscreen[MAXREAD];        /* screen command during err  */
500
501         char  smode[NAMESZ];               /* current  expected result   */
502         char  emode[NAMESZ];               /* previous expected result   */
503
504         char  sname[NAMESZ];               /* current rc script name     */
505         char  ename[NAMESZ];               /* rc script name during err  */
506         char  tname[TIDSZ];           /* current rc script test id# */
507
508         char  inpkg[80];              /* input pkg string           */
509         char  linpkg[80];             /* last input pkg string      */
510
511         char  pack_name[40];               /* input pkg being analyzed   */
512         char  lpack_name[40];              /* last  pkg being analyzed   */
513
514         char  oper[OPERSZ+2];              /* current Form operation     */
515
516         char  tbuf[MAXREAD];               /* temporary buffer           */
517
518         char  thdr0[RPTWIDTH];             /* temporary header buffer    */
519         char  thdr3[RPTWIDTH];             /* temporary header buffer    */
520
521         char  temp_name[NAMESZ];      /* temporary name for heap    */
522         char  key_name[NAMESZ];            /* temporary name for heap    */
523         char  time_buf[40];             /* time string buffer         */
524
525         char  outfile[80];              /* RC output file name        */
526         char  prefix[MAXPFX];                /* prefix derrived from pkg   */
527
528         char  file_Lpfx[MAXPFX];        /* Lower prefix to search for */
529         char  file_Upfx[MAXPFX];        /* Upper prefix to search for */
530         char  file_sfx[MAXPFX];              /* suffix to search for       */
531         char  last_errf[60];                 /* previous package error file*/
532         char  caption[CAPLIMIT+1];           /* caption for type of tests  */
533         char  usrgen[GENLIMIT+1];            /* package generic i.e. 5e11.1*/
534         char  loadid;                   /* package loadid  i.e. z     */
535
536
537         char  tmpnum[15];             /* temporary number buffer    */
```

```
538
539         char *timestr;                    /* ctime() returned string    */
540         char *strptr1;              /* string pointer             */
541
542         short first_time;           /* start of loop indication   */
543
544         short foundForm;            /* rc found Form flag         */
545         short rc_done;              /* rc operation has completed */
546         short rc_err;               /* rc error flag              */
547         short done_read;            /* lines read since last rc   */
548
549         short su_package;           /* flag saying its an su pkg  */
550         short diff;                 /* flag saying to diff 2 pkgs */
551
552         short dotno;                /* loop counter               */
553         short non_numeric;          /* non numeric field flag     */
554         short errind;               /* error indication flag      */
555         short uclass;               /* class array index to update*/
556         short arg_no;               /* input file to process argno*/
557         short pkgflag;              /* prev or curr package flag  */
558         short off;                  /* offset from the beginning  */
559                                 /* Inbuf where the error      */
560                                 /* starts                     */
561
562         int   last_mbr;                   /* last member for heap       */
563         int   great_son;            /* great son   for heap       */
564         int   parent;                  /* parent      for heap       */
565         int   temp_count;                 /* temp count  for heap       */
566         int   key_count;            /* key  count  for heap       */
567
568         int   i;                    /* index variable             */
569         int   j;                    /* index variable             */
570         int   k;                    /* index variable             */
571         int   m;                    /* index variable             */
572         int   pos;                  /* index variable             */
573         int   loc;                  /* index variable             */
574
575         int   errstart;                   /* position within Inbuf      */
576         int   errstop;              /* position within Inbuf      */
```

```
577         int   lineno;               /* position within Inbuf      */
578
579         int   indxlen;              /* length of index string     */
580         int   indxnum;              /* index number               */
581         int   newcnt;               /* count of new errors in pkg */
582
583         int   pageno;               /* page number display        */
584         int   pos1end;              /* offset to print at col. 2  */
585         int   to_prt;               /* number of lines to print   */
586
587         long  secs;                 /* used in time() call        */
588         float passper;              /* pass percentage            */
589         float failper;              /* fail percentage            */
590
591         DIR     *src_dfp;           /* source data file pointer   */
592         struct   dirent *readdir();       /* reads directory entries    */
593         register struct dirent *src_dp;
594
595
596
597
598         /* BEGIN EXECUTABLE CODE */
599
600         /* Unbuffer stderr and stdout */
601
602         setbuf(stderr, (char *) NULL);
603         setbuf(stdout, (char *) NULL);
604
605         /* Capture the command line arguments */
606
607         errind       = FALSE;
608         diff         = FALSE;
609         su_package   = FALSE;
610         Vlog         = FALSE;
611         Slog         = FALSE;
612         Olog         = FALSE;
613         pack_name[0] = '\0';
614         caption[0]   = '\0';
615         usrgen[0]    = '\0';
```

```
616
617         while ((i = getopt(argc, argv, "vsodc:g:")) != EOF)
618         {
619             switch(i)
620             {
621                 case 'c':
622                 {
623                     /* Get a users caption - maximum of 26 characters */
624
625                     (void) strcpy(caption, optarg);
626                     caption[CAPLIMIT] = '\0';
627
628                     break;
629                 }
630
631                 case 'v':
632                 {
633                     /* Log errors by view number */
634
635                     Vlog = TRUE;
636                     break;
637                 }
638
639                 case 's':
640                 {
641                     /* Log errors by script name */
642
643                     Slog = TRUE;
644                     break;
645                 }
646
647                 case 'o':
648                 {
649                     /* Log errors in output file */
650
651                     Olog = TRUE;
652                     break;
653                 }
654
```

```
655             case 'd':
656             {
657                 /* Perform a diff on two packages */
658 
659                 diff = TRUE;
660                 break;
661             }
662 
663             case 'g':
664             {
665                 /* User has a generic to appear in caption */
666 
667                 (void) strcpy(usrgen, optarg);
668                 caption[GENLIMIT] = '\0';
669                 break;
670             }
671 
672             case '?':
673             default :
674             {
675                 errind = TRUE;
676                 break;
677             }
678 
679         }
680     }
681 
682     /* Validate options */
683 
684     if ( (Vlog == TRUE) && (Slog == TRUE) || (optind == argc) )
685     {
686         /* 1. User can not select error logging by */
687         /*    both view number and script name.    */
688 
689         /* 2. No input files were specified        */
690 
691         errind = TRUE;
692     }
693 
```

```
694         if ( (Slog == FALSE) && (Vlog == FALSE) )
695         {
696             /* No options for logging were set - set default */
697
698             Olog = TRUE;
699         }
700
701
702         /*********************************/
703         /* Validate all input files exist */
704         /*********************************/
705
706         if (errind != TRUE)
707         {
708             Runhist.start[0]    = '\0';
709             Runhist.end[0]      = '\0';
710
711             for (i=optind; i<argc; i++)
712             {
713                 /*******************************************/
714                 /* Continually loop through the history file */
715                 /*******************************************/
716
717                 /* Find out if a history file exists for the current run */
718                 /* If it does, we can calculate run duration time        */
719
720                 if ((pos = sindex(argv[i], DOTRCDOT)) != NOSTRFOUND)
721                 {
722                     /* create the history file name as package.HIST.suffix */
723
724                     (void) strcpy(Runhist.filename, argv[i]);
725                     (void) strcpy(&Runhist.filename[pos - 1], DOTHISTDOT);
726                     (void) strcat(Runhist.filename, &argv[i][pos + 3]);
727                 }
728
729                 if ((Hfp = fopen(Runhist.filename, "r")) != NULL)
730                 {
731                     while (TRUE)
732                     {
```

```
                    if ((strptr1 = fgets(Inbuf[0], 79, Hfp)) == NULL)
                    {
                        break;
                    }

                    if (Runhist.start[0] == '\0')
                    {
                        (void) strcpy(Runhist.start, Inbuf[0]);
                    }
                }
                (void) strcpy(Runhist.end, Inbuf[0]);
                (void) fclose(Hfp);
            }

            /* INPUT File name */

            if ((Ifp = fopen(argv[i], "r")) == NULL)
            {
                (void) fprintf(stderr,
                                "\nError: Can't open input file : %s\n",
                                 argv[i]);
                errind = TRUE;
                break;
            }
            else
            {
                (void) fclose(Ifp);
            }
        }
    }


    if (errind == TRUE)
    {
        if ((strptr1 = strrchr(argv[0], '/')) != NULL)
        {
            (void) strcpy(tbuf, strptr1+1);
        }
        else
```

```
772         {
773             (void) strcpy(tbuf, argv[0]);
774 
775         }
776         (void) fprintf(stderr, "%s: Software Version: %s %s\n",
777                tbuf, Version, Delta);
778         (void) fprintf(stderr,
779                "\nUsage:  %s [-c 'caption' ] [-v | -s] [-o] [-d] RCoutfile  [RCoutfile...]\n",
780                        tbuf);
781 
782         (void) fprintf(stderr,
783                "\nwhere     c - user defined caption (%d character limit)\n",
784                CAPLIMIT);
785         (void) fprintf(stderr,
786                "          v - log errors by view number\n");
787 
788         (void) fprintf(stderr,
789                "          s - log errors by script name\n");
790 
791         (void) fprintf(stderr,
792                "          o - log errors in the output file (DEFAULT)\n");
793 
794         (void) fprintf(stderr,
795                "          d - attempts to diff a previous package against this one\n");
796 
797         (void) fprintf(stderr,
798                "\nExample: %s -c 'Stability' -s 3m00.RC.1\n\n", tbuf);
799 
800         (void) fprintf(stderr,
801                "NOTE: Package Name is derrived from the first four characters of\n");
802         (void) fprintf(stderr,
803                "      the RCoutfile name.  Output file names should have a suffix of: \n");
804         (void) fprintf(stderr,
805                "\n     RC.x  -  where x is some number\n\n");
806 
807         exit(1);
808     }
809     /*********************************************/
810     /* Determine the start and stop time of the run */
```

```
811         /***********************************************/
812
813         if ((Runhist.start[0] != '\0') && (Runhist.end[0] != '\0'))
814         {
815             /* strip script file name from prefix and stop time suffix */
816
817             if ((pos = sindex(Runhist.start, SPPIPESP)) != NOSTRFOUND)
818             {
819                 (void) strcpy(Inbuf[0], &Runhist.start[pos + 2]);
820                 if ((pos = sindex(Inbuf[0], SPPIPESP)) != NOSTRFOUND)
821                 {
822                     Inbuf[0][pos - 1] = '\0';
823                     (void) strcpy(Runhist.start, Inbuf[0]);
824                 }
825
826             }
827             /* strip script file name and start time prefix */
828
829             if ((pos = sindex(Runhist.end, SPPIPESP)) != NOSTRFOUND)
830             {
831                 (void) strcpy(Inbuf[0], &Runhist.end[pos + 2]);
832                 if ((pos = sindex(Inbuf[0], SPPIPESP)) != NOSTRFOUND)
833                 {
834                     (void) strcpy(Runhist.end, &Inbuf[0][pos + 2]);
835
836                     /* strip off the new line character */
837
838                     Runhist.end[strlen(Runhist.end)-1] = '\0';
839                 }
840
841             }
842
843             Runhist.start_day[0]  = '\0';
844             Runhist.start_time[0] = '\0';
845             Runhist.end_day[0]    = '\0';
846             Runhist.end_time[0]   = '\0';
847
848             for (i=0; i<strlen(Runhist.start); i++)
849             {
```

```
850             if (Runhist.start[i] == ' ')
851             {
852                (void) strncpy(Runhist.start_day,  Runhist.start, i);
853                (void) strcpy (Runhist.start_time, &Runhist.start[i+1]);
854                break;
855             }
856          }
857
858          for (i=0; i<strlen(Runhist.end); i++)
859          {
860             if (Runhist.end[i] == ' ')
861             {
862                (void) strncpy(Runhist.end_day,   Runhist.end, i);
863                (void) strcpy (Runhist.end_time, &Runhist.end[i+1]);
864                break;
865             }
866          }
867
868          /* Determine the integer values for the character strings */
869
870           /* STARTING DAY */
871
872          Runhist.start_day[2] = '\0';
873          Runhist.start_day[5] = '\0';
874
875          Runhist.smon = atoi( Runhist.start_day);
876          Runhist.sday = atoi(&Runhist.start_day[3]);
877          Runhist.syr  = atoi(&Runhist.start_day[6]);
878
879          Runhist.start_day[2] = '/';
880          Runhist.start_day[5] = '/';
881
882           /* ENDING DAY */
883
884          Runhist.end_day[2] = '\0';
885          Runhist.end_day[5] = '\0';
886
887          Runhist.emon = atoi( Runhist.end_day);
888          Runhist.eday = atoi(&Runhist.end_day[3]);
```

```
889         Runhist.eyr  = atoi(&Runhist.end_day[6]);
890
891         Runhist.end_day[2] = '/';
892         Runhist.end_day[5] = '/';
893
894          /* STARTING TIME */
895
896         Runhist.start_time[2] = '\0';
897         Runhist.start_time[5] = '\0';
898
899         Runhist.shr   = atoi( Runhist.start_time);
900         Runhist.smin  = atoi(&Runhist.start_time[3]);
901         Runhist.ssec  = atoi(&Runhist.start_time[6]);
902
903         Runhist.start_time[2] = ':';
904         Runhist.start_time[5] = ':';
905
906          /* ENDING TIME */
907
908         Runhist.end_time[2] = '\0';
909         Runhist.end_time[5] = '\0';
910
911         Runhist.ehr  = atoi( Runhist.end_time);
912         Runhist.emin = atoi(&Runhist.end_time[3]);
913         Runhist.esec = atoi(&Runhist.end_time[6]);
914
915         Runhist.end_time[2] = ':';
916         Runhist.end_time[5] = ':';
917
918         if ((Runhist.smon == Runhist.emon) &&
919             (Runhist.sday == Runhist.eday))
920         {
921             /* Run started and completed on the same day.   */
922             /* Now determine the run duration time          */
923
924             Runhist.start_sec  = Runhist.shr * HRSEC;
925             Runhist.start_sec += (Runhist.smin * MINSEC);
926             Runhist.start_sec += Runhist.ssec;
927
```

```
928             Runhist.end_sec  = Runhist.ehr * HRSEC;
929             Runhist.end_sec += (Runhist.emin * MINSEC);
930             Runhist.end_sec += Runhist.esec;
931
932             Runhist.run_sec = Runhist.end_sec - Runhist.start_sec;
933         }
934         else
935         {
936             /* Run started and completed on different days. */
937             /* Now determine the run duration time          */
938
939             /* Determine the run duration seconds for the first day */
940
941             Runhist.start_sec  = Runhist.shr * HRSEC;
942             Runhist.start_sec += (Runhist.smin * MINSEC);
943             Runhist.start_sec += Runhist.ssec;
944
945             Runhist.start_sec = DAYSEC - Runhist.start_sec;
946
947             /* Now keep incrementing the day count until you reach */
948             /* the date the run completed.                         */
949
950             i = Runhist.smon;
951             j = Runhist.sday;
952
953             /* Determine if this is a leap year -- if so set k = 1 */
954
955             if (Runhist.syr == 0)
956                 k = 1;
957             else
958                 if (Runhist.syr < 4)
959                     k = 0;
960                 else
961                     if ((Runhist.syr % 4) == 0)
962                         k = 1;
963                      else
964                         k = 0;
965
966             Runhist.run_sec = 0L;
```

```
967             while (1)
968             {
969                 j++;
970
971                 if (j == Runhist.eday)
972                 {
973                     /* Determine the run duration seconds for the */
974                     /* last day                                   */
975
976                     Runhist.end_sec  = Runhist.ehr * HRSEC;
977                     Runhist.end_sec += (Runhist.emin * MINSEC);
978                     Runhist.end_sec += Runhist.esec;
979
980                     /* Add up all the seconds consumed */
981
982                     Runhist.run_sec += Runhist.start_sec + Runhist.end_sec;
983                     break;
984                 }
985                 else
986                 {
987                     /* Did we exceed the maximum days for this month? */
988
989                  if ((i == 1) || (i == 3)  || (i == 5)  || (i == 7) ||
990                      (i == 8) || (i == 10) || (i == 12) )
991                  {
992                         Runhist.days = 31;
993                  }
994                  else
995                  {
996                        if (i == 2)
997                        {
998                             Runhist.days = 28 + k;
999                        }
1000                       else
1001                       {
1002                            Runhist.days = 30;
1003                       }
1004                 }
1005
```

```
006                     if (j > Runhist.days)
007                     {
008                         j = 0;
009                         i++;
010                         if (i > 12)
011                             i = 1;
012                     }
013                     else
014                     {
015                         Runhist.run_sec += DAYSEC;
016                     }
017                 }
018
019             }   /* end of while (1) */
020         }
021
022         /*****************************/
023         /* CALCULATE THE RUN DURATION */
024         /*****************************/
025
026         Runhist.days    = (Runhist.run_sec / DAYSEC);
027         Runhist.remain  = (Runhist.run_sec % DAYSEC);
028
029         Runhist.hours   = (Runhist.remain  / HRSEC);
030         Runhist.remain  = (Runhist.remain  % HRSEC);
031
032         Runhist.mins    = (Runhist.remain  / MINSEC);
033         Runhist.remain  = (Runhist.remain  % MINSEC);
034
035         Runhist.secs    = Runhist.remain;
036     }
037
038     /* Derrive Package Name From output Name Given By User   */
039
040
041     if ((strptr1 = strrchr(argv[optind], '/')) == NULL)
042     {
043          /* No slashes found */
044
```

```
045          (void) strcpy(outfile, argv[optind]);
046       }
047       else
048       {
049          /* Slash(es) were specified - point to file name */
050
051          (void) strcpy(outfile, strptr1+1);
052       }
053
054       if ((pos = sindex(outfile, DOTRCDOT)) != NOSTRFOUND)
055       {
056           pos--;
057           if (pos > MAXPFX-1)
058               pos = MAXPFX-1;
059
060           (void) strncpy(pack_name, outfile, pos);
061           pack_name[pos] = '\0';
062       }
063       else
064       {
065           (void) strncpy(pack_name, outfile, MAXPFX-1);
066           pack_name[MAXPFX-1] = '\0';
067       }
068
069       /****************************************/
070       /* Determine The loadid Of The Of Package */
071       /****************************************/
072
073       loadid = '?';
074
075       if ( (((pack_name[0]) == '8')   || ((pack_name[0]) == '9')    ||
076             ((pack_name[0]) == 'm')   || ((pack_name[0]) == 'a')    ||
077             ((pack_name[0]) == 'z') ) && ((strlen(pack_name)) == 5) )
078       {
079           /************************/
080           /* Official BWM Package */
081           /************************/
082
083           su_package = TRUE;
```

```
084             loadid = pack_name[0];
085         }
086
087         if ((strlen(pack_name)) == 4)
088            loadid = pack_name[1];
089
090         if ( ((pack_name[0]) == '3') && ((strlen(pack_name)) == 4) )
091         {
092             /****************/
093             /* 3000 Package */
094             /****************/
095
096             su_package = TRUE;
097             loadid = pack_name[1];
098         }
099
100
101         /***********************************************/
102         /* Determine The GENERIC Of The Type Of Package */
103         /***********************************************/
104
105         if (usrgen[0] == '\0')
106         {
107             if ( ((strncmp(pack_name, "bwm", 3)) == 0) ||
108                  ((strncmp(pack_name, "BWM", 3)) == 0) ||
109                  ((strncmp(pack_name, "CFT", 3)) == 0) )
110
111             {
112                 /************************/
113                 /* Official Package     */
114                 /************************/
115
116                 su_package = TRUE;
117
118                 /* Look for the minus seperator in bwm9x-xxxx */
119                 if ((pos = sindex(pack_name, MINUS)) != NOSTRFOUND)
120                 {
121                     (void) strncpy(prefix, &pack_name[pos], MAXPFX-1);
122                     pack_name[MAXPFX-1] = '\0';
```

```
123             }
124             else
125             {
126                 /* Couldn't find minus separator */
127
128                 pos = strlen(pack_name);
129                 if (pos > 4)
130                 {
131                     /* Copy the last 4 characters */
132
133                     pos = pos - 4;
134                     (void) strncpy(prefix, &pack_name[pos], MAXPFX-1);
135                     pack_name[MAXPFX-1] = '\0';
136                 }
137                 else
138                 {
139                     /* Copy what you got */
140
141                     (void) strcpy(prefix, pack_name);
142                 }
143             }
144
145             /* Adjust the pack name accordingly ... */
146
147             (void) strcpy(pack_name, prefix);
148         }
149         else
150         {
151             /* Determine the user generic */
152
153             switch(loadid)
154             {
155                 case '8':
156                 {
157                     (void) strcpy(usrgen, "5e8.1");
158                     break;
159                 }
160                 case '9':
161                 {
```

```
                    (void) strcpy(usrgen, "5e9.1");
                    break;
                }
                case 'm':
                case 'M':
                {
                    (void) strcpy(usrgen, "5e9.2");
                    break;
                }
                case 'a':
                case 'A':
                {
                    (void) strcpy(usrgen, "5e10.1");
                    break;
                }
                case 'z':
                case 'Z':
                {
                    (void) strcpy(usrgen, "5e11.1");
                    break;
                }
                case 'i':
                case 'I':
                {
                    (void) strcpy(usrgen, "5e12.1");
                    break;
                }

                default:
                {
                    (void) strcpy(usrgen, "5e??.?");
                    break;
                }
            }
        }
    }

    (void) sprintf(Rpt_file, "%s%s", pack_name, RPTSUFFIX);
    if ((Rfp = fopen(Rpt_file, "w")) == NULL)
```

```
201         {
202             (void) fprintf(stderr,
203                             "\nError: Can't open report file: %s\n", Rpt_file);
204             (void) fclose(Ifp);
205             exit(1);
206         }
207
208         (void) sprintf(Out_file, "%s%s", pack_name, OUTSUFFIX);
209         if ((Ofp = fopen(Out_file, "w")) == NULL)
210         {
211             (void) fprintf(stderr,
212                             "\nError: Can't open output file: %s\n", Out_file);
213             (void) fclose(Ifp);
214             (void) fclose(Rfp);
215             exit(1);
216         }
217         (void) sprintf(Err_file, "%s%s", pack_name, ERRSUFFIX);
218
219         /* Initialize program initial defaults */
220
221         Tlines          = 0L;
222         Tstfail         = 0L;
223         Tstpass         = 0L;
224         Pnew            = 0L;
225         Pchg            = 0L;
226         Pout            = 0L;
227         Pvfy            = 0L;
228         Pmvfy           = 0L;
229         Pshvfy          = 0L;
230         Pattr           = 0L;
231         Punkwn          = 0L;
232         Fnew            = 0L;
233         Fchg            = 0L;
234         Fout            = 0L;
235         Fvfy            = 0L;
236         Fmvfy           = 0L;
237         Fshvfy          = 0L;
238         Fattr           = 0L;
239         Funkwn          = 0L;
```

```
240         lineno          = 0;
241         dotno           = 0;
242         uclass          = 0;
243         Serr_cnt        = 0;
244         done_read       = 0;
245
246         Errdict.entries = 0;
247         Nperr.entries   = 0;
248         Operr.entries   = 0;
249
250         /* Initialize RC/V Form Error Stats */
251
252         for (i=0; i<MAXCLASS; i++)
253             for (j=0; j<MAXVIEW; j++)
254             {
255                 Class[i].View[j].Pass.new   = 0;
256                 Class[i].View[j].Pass.chg   = 0;
257                 Class[i].View[j].Pass.out   = 0;
258                 Class[i].View[j].Pass.vfy   = 0;
259                 Class[i].View[j].Pass.mvfy  = 0;
260                 Class[i].View[j].Pass.shvfy = 0;
261                 Class[i].View[j].Pass.attr  = 0;
262                 Class[i].View[j].Pass.unkwn = 0;
263                 Class[i].View[j].Pass.count = 0;
264
265                 Class[i].View[j].Fail.new   = 0;
266                 Class[i].View[j].Fail.chg   = 0;
267                 Class[i].View[j].Fail.out   = 0;
268                 Class[i].View[j].Fail.vfy   = 0;
269                 Class[i].View[j].Fail.mvfy  = 0;
270                 Class[i].View[j].Fail.shvfy = 0;
271                 Class[i].View[j].Fail.attr  = 0;
272                 Class[i].View[j].Fail.unkwn = 0;
273                 Class[i].View[j].Fail.count = 0;
274             }
275
276
277         for (i=0; i<MAXSCRERR; i++)
278         {
```

```
279         Rc[j].name[0] = '\0';
280         Rc[j].count   = 0;
281     }
282
283
284     /*****************************************************/
285     /* CAPTURE THE CURRENT DATE AND TIME FOR THE REPORT */
286     /*****************************************************/
287
288     secs    = 0L;
289        secs    = time(&secs);
290        timestr = ctime(&secs);
291
292        (void) strcpy(time_buf, timestr);
293        time_buf[strlen(time_buf)-1] = '\0';
294
295     /**************************/
296     /* PROCESS EACH INPUT FILE */
297     /**************************/
298
299     for (arg_no = optind; arg_no < argc; arg_no++)
300     {
301         first_time   = TRUE;
302         rc_err       = FALSE;
303         rc_done      = FALSE;
304         foundForm    = FALSE;
305         Linesread    = 0L;
306         tname[0]     = '\0';
307
308         (void) strcpy(sname, "UNKNOWN");
309         (void) strcpy(smode, SPASS);
310         (void) sprintf(lastscreen,
311                        "%-10ld: %s", 0L, " #screen PASS - DEFAULT");
312
313         if (arg_no != optind)
314         {
315             (void) fprintf(stdout,
316                            "\n\nRCV ANALYSIS COMPLETE FOR: %s \n", inpkg);
317             (void) fclose(Ifp);
```

```
318         }
319
320         if ((Ifp = fopen(argv[arg_no], "r")) == NULL)
321         {
322             (void) fprintf(stderr,
323                            "\nFATAL SYSTEM ERROR: Can't re-open input file : %s\n",
324                             argv[arg_no]);
325             exit(1);
326         }
327
328         /* Strip off the slashes */
329
330         if ((strptr1 = strrchr(argv[arg_no], '/')) == NULL)
331         {
332              /* No slashes found */
333
334              (void) strcpy(tbuf, argv[arg_no]);
335         }
336         else
337         {
338              /* Slash(es) were specified - point to file name */
339
340              (void) strcpy(tbuf, strptr1+1);
341         }
342
343         (void) sprintf(inpkg, "%s : %s",
344                        pack_name, tbuf);
345
346         (void) fprintf(stdout,
347                        "\n\nANALYSIS STARTING FOR: %s\n\n", inpkg);
348
349         while (TRUE)
350         {
351             /*************************************************/
352             /* CONTINUALLY READ THE INPUT Application FILE   */
353             /*************************************************/
354
355             /* Skip the first 12 positions in the buffer - allow for 10   */
356             /* position long integer followed by a colon and a blank.     */
```

```
357
358             Linesread++;
359             Tlines++;
360             (void) sprintf(Inbuf[lineno], "%10ld: ", Linesread);
361
362             if ((strptr1 = fgets(&Inbuf[lineno][12], MAXREAD-1, Ifp)) == NULL)
363             {
364                 Linesread--;
365                 Tlines--;
366                 break;
367             }
368
369             /* Print a dot ever 1000 operations */
370             dotno++;
371             if (dotno == 1000)
372             {
373                 (void) fprintf(stdout, ".");
374                 dotno = 0;
375             }
376 #ifdef DEBUG
377             printf("Inbuf[%3d]: %s", lineno, Inbuf[lineno]);
378 #endif
379
380             /* Transform input buffer to all lower case */
381
382             for (i=0; i<strlen(Inbuf[lineno]); i++)
383             {
384                 tbuf[i] = tolower(Inbuf[lineno][i]);
385             }
386             tbuf[i] = '\0';
387
388
389             /************************************/
390             /* DETERMINE WHAT THE INPUT LINE IS */
391             /************************************/
392
393 parse_again:
394
395             if ((pos = sindex(tbuf, SCRIPTNM)) != NOSTRFOUND)
```

```
{
    /***************/
    /* SCRIPT NAME */
    /***************/

    /* Clear the test script identifier */
    tname[0] = '\0';

    /* Use the Inbuf array vs. tbuf because tbuf was          */
    /* converted to lower case...                             */


    /* Remove the new line character from the input line      */

    pos = strlen(Inbuf[lineno]);
    Inbuf[lineno][pos-1] = '\0';


    /* Look for the last slash in the script name (in case of */
    /* full path script names)                                */

    if ((strptr1 = strrchr(Inbuf[lineno], '/')) != NULL)
    {
        /* FULL PATH TO SCRIPT NAME WAS SPECIFIED */

        (void) strncpy(sname, strptr1+1, NAMESZ-1);
    }
    else
    {
        /* FULL PATH TO SCRIPT NAME WAS NOT SPECIFIED */

        /* Look for the word 'file' within the input string.  */
        /* Set pos to character postiion within the string    */
        /* containing the word 'file':                        */
        /*                                                    */
        /*      "Application: using input file xxxxx"         */
        /*                                                    */
        /* Note: pos-1 is the actual C array position         */
        /*                                                    */
```

```
435
436                         pos = sindex(Inbuf[lineno], "file");
437
438                         (void) strncpy(sname, &Inbuf[lineno][pos+4], NAMESZ-1);
439                     }
440
441                     (void) strcpy(smode, SPASS);
442                     (void) sprintf(lastscreen,
443                                 "%-10ld: %s", 0L, " #screen PASS - DEFAULT");
444
445                     /* Throw away script name line */
446
447                     continue;
448                 }
449
450                 if ((pos = sindex(tbuf, STARTTID)) != NOSTRFOUND)
451                 {
452                     /*****************************/
453                     /* STARTTID - test id number */
454                     /*****************************/
455
456                     if ((pos = sindex(tbuf, TIDPFX)) != NOSTRFOUND)
457                     {
458                        i =  0;
459                        for (j=pos - 1;
460                             tbuf[j] != ' ' && tbuf[j] != '\n' && i < TIDSZ;
461                             j++)
462                        {
463                            tname[i] = tbuf[j];
464                            i++;
465                        }
466                        tname[i] = '\0';
467                     }
468                 }
469
470
471                 if ((pos = sindex(tbuf, SCREEN)) != NOSTRFOUND)
472                 {
473                     /***************/
```

```
474             /* #SCREEN ... */
475             /***************/
476
477             /* Validate the mode received */
478
479             if ((pos = sindex(Inbuf[lineno], SPASS)) != NOSTRFOUND)
480             {
481                 (void) strcpy(smode, SPASS);
482             }
483             else if ((pos = sindex(Inbuf[lineno], SFAIL)) != NOSTRFOUND)
484             {
485                 (void) strcpy(smode, SFAIL);
486             }
487             else if ((pos = sindex(Inbuf[lineno], SNOCARE)) != NOSTRFOUND)
488             {
489                 (void) strcpy(smode, SNOCARE);
490             }
491             else
492             {
493                 (void) fprintf(stderr,
494                                "Error: Unknown screen command: [%s]\n",
495                                tbuf);
496             }
497             (void) sprintf(lastscreen, "%s", Inbuf[lineno]);
498
499             /* Throw away screen line */
500             continue;
501         }
502
503
504         /* Search for form= lines and ignore comment lines */
505
506         if ((pos = sindex(tbuf, LBFORM)) != NOSTRFOUND)
507         {
508             /* Throw away screen line */
509
510             continue;
511         }
512
```

```
513             else if ((pos  = sindex(tbuf, FORM)) != NOSTRFOUND)
514             {
515                 /************/
516                 /* FORM=... */
517                 /************/
518
519
520                 /*****************************************/
521                 /* UPDATE THE STATS FROM THE PREVIOUS RC */
522                 /*****************************************/
523
524                 if (first_time != TRUE)
525                 {
526
527                     /* Let A views occupy 0th element of array.           */
528                     /* Let B views occupy 1st element of array.           */
529                     /* Let other views occupy +2 offset element of array. */
530
531                     uclass = Cur_classnum;
532                     if (uclass == 0)
533                     {
534                         /* uclass is zero when Cur_classnum encountered a */
535                         /* non-numeric number -- classes A and B          */
536
537                         if (Cur_class[0] == 'b')
538                             uclass++;
539                     }
540                     else
541                     {
542                         uclass = uclass + 2;
543                         if (uclass >= MAXCLASS)
544                             uclass = MAXCLASS - 1;
545                     }
546
547
548                     if (rc_err == TRUE)
549                     {
550                         /***********************************************/
551                         /* FIND THE START AND END OF THE ERROR MESSAGES */
```

```
552                     /***********************************************/
553
554                     for (i=0; i<lineno; i++)
555                     {
556                        if ((loc = sindex(Inbuf[i], MSGU)) != NOSTRFOUND)
557                            break;
558                     }
559
560                     if (i == lineno)
561                        {
562                         /* Look for 1st Application: a FAIL was expected */
563                         /* Expected fails will have this line            */
564
565                         for (i=0; i<lineno; i++)
566                         {
567                            if ((loc = sindex(Inbuf[i], MEXFAIL)) != NOSTRFOUND)
568                                break;
569                         }
570                         if (i == lineno)
571                         {
572
573                             (void) fprintf(stderr,
574                             "\nError:  Can't find [%s or %s] error line\n",
575                             MSGU, MEXFAIL);
576
577                             (void) fprintf(stderr, "BUFFER CONTAINS -\n");
578
579                             for (j=0; j<=lineno; j++)
580                                 (void) fprintf(stderr,
581                                        "Inbuf[%d]: %s", j, Inbuf[j]);
582                             (void) fprintf(stderr, "\n");
583
584                                exit(1);
585                         }
586                         else
587                         {
588                             errstart = errstop = i - 1;
589                         }
590                        }
```

```
591                     else
592                        {
593                         errstart = i;
594
595                         /*********************************/
596                         /* FIND ?E no more error messages */
597                         /*********************************/
598
599                         for (j=errstart; j<lineno; j++)
600                         {
601                            if ((loc = sindex(Inbuf[j], MNOMORE)) != NOSTRFOUND)
602                                break;
603                         }
604
605                         if (j == lineno)
606                            {
607                             (void) fprintf(stderr,
608                             "\nError:  Can't find [%s] error line\n",
609                             MNOMORE);
610
611                             (void) fprintf(stderr, "BUFFER CONTAINS -\n");
612
613                             for (k=0; k<=lineno; k++)
614                                 (void) fprintf(stderr,
615                                        "Inbuf[%d]: %s", k, Inbuf[k]);
616                             (void) fprintf(stderr, "\n");
617
618                                exit(1);
619                            }
620
621                         errstop = j - 1;
622                        }
623
624                     /*****************************/
625                     /* PROCESS THE ERROR MESSAGE */
626                     /*****************************/
627
628                     prcs_err(lineno-1, emode, uclass, Cur_viewnum,
629                              ename, tname, inpkg, elastscreen,
```

```
                    errstart, errstop);

         /* Update the stats of the last error script */

            updstat(rc_err, uclass, Cur_viewnum, lineno, oper,
                      ename);
        }
        else
        {
         /* Update the stats of the last good script */

            updstat(rc_err, uclass, Cur_viewnum, lineno, oper,
                      sname);
        }
    }

    /* Move the current form to the first position */
    /* in the input buffer.                        */

    if (lineno > 0 )
    {
        (void) strcpy(Inbuf[0], Inbuf[lineno]);
        lineno = 0;
    }

    first_time = FALSE;

    foundForm  = TRUE;
    rc_err     = FALSE;
    rc_done    = FALSE;

    /***********************/
    /* Determine the Class */
    /***********************/

    j           = 0;
    non_numeric = FALSE;

    /* Look for v as the  seperator                */
```

```
669                 /* Note: pos is set to the blank before 'form=' */
670
671                 for (i=pos+5; tbuf[i] != 'v' && j < CLASSSZ; i++)
672                 {
673                     /* Skip blanks */
674
675                     if (tbuf[i] != ' ')
676                     {
677
678                         if ((tbuf[i] < '0') || (tbuf[i] > '9'))
679                             non_numeric = TRUE;
680
681                         Cur_class[j] = tbuf[i];
682                         j++;
683                     }
684                 }
685                 Cur_class[j] = '\0';
686
687                 if (j == 0)
688                     non_numeric = TRUE;
689
690                 if (non_numeric == FALSE)
691                 {
692                      Cur_classnum = atoi(Cur_class);
693                 }
694                 else
695                 {
696                      Cur_classnum = 0;
697                 }
698
699                 if (Cur_classnum >= TOTCLASS)
700                 {
701                     (void) fprintf(stderr,
702                            "\nError: Class number: %d exceeds maximum array size\n",
703                                    Cur_classnum);
704
705                     (void) fprintf(stderr,
706                            "LAST LINE READ WAS -\n%s", Inbuf[lineno]);
707
```

```
708                 Cur_classnum = MAXCLASS - 1;
709                 Cur_viewnum  = MAXVIEW  - 1;
710                 (void) strcpy(oper, UNKWN );
711
712                 (void) fprintf(stderr,
713                               "Assigning failure to View: %d.%d  Operation: %s\n",
714                                Cur_classnum, Cur_viewnum, oper);
715             }
716             else
717             {
718                 /**********************/
719                 /* Determine the View */
720                 /**********************/
721
722                 k           = 0;
723                 non_numeric = FALSE;
724
725                 /* Look for the ampersand as the seperator */
726
727                 for (j=i+1; tbuf[j] != '&' && k < VIEWSZ; j++)
728                 {
729                     if ((tbuf[j] < '0') || (tbuf[j] > '9'))
730                         non_numeric = TRUE;
731
732                     Cur_view[k] = tbuf[j];
733                     k++;
734                 }
735                 Cur_view[k] = '\0';
736
737                 if (k == 0)
738                     non_numeric = TRUE;
739
740                 if (non_numeric == FALSE)
741                 {
742                      Cur_viewnum = atoi(Cur_view);
743                 }
744                 else
745                 {
746                      Cur_viewnum = 0;
```

```
747                 }
748
749                 if (Cur_viewnum >= TOTVIEW)
750                 {
751                     (void) fprintf(stderr,
752                           "\nError: View number: %d exceeds maximum array size\n",
753                            Cur_viewnum);
754
755                     (void) fprintf(stderr,
756                              "LAST LINE READ WAS -\n%s", Inbuf[lineno]);
757
758                     Cur_classnum = MAXCLASS - 1;
759                     Cur_viewnum  = MAXVIEW  - 1;
760                     (void) strcpy(oper, UNKWN);
761
762                     (void) fprintf(stderr,
763                              "Assigning failure to View: %d.%d  Operation: %s\n",
764                               Cur_classnum, Cur_viewnum, oper);
765                 }
766                 else
767                 {
768                     /***************************/
769                     /* DETERMINE THE OPERATION */
770                     /***************************/
771
772                     /* Allow for comma (i.e. single line apptext      */
773                     /* input                                          */
774
775                     i = 0;
776                     for (k=j+1; tbuf[k] != '!' &&
777                                    tbuf[k] != ' ' &&
778                                 tbuf[k] != ',' &&
779                                 i < OPERSZ; k++)
780                     {
781                         oper[i] = tbuf[k];
782                         i++;
783                     }
784
785                     if (i == 0)
```

```
{
    (void) strcpy(oper, UNKWN);
}
else
{
    oper[i]   = '!';
    oper[i+1] = '\0';

    /* Validate operation */

    if (((i = sindex(oper, NEW))   != NOSTRFOUND) &&
        ((i = sindex(oper, CHG))   != NOSTRFOUND) &&
        ((i = sindex(oper, OUT))   != NOSTRFOUND) &&
        ((i = sindex(oper, VFY))   != NOSTRFOUND) &&
        ((i = sindex(oper, MVFY))  != NOSTRFOUND) &&
        ((i = sindex(oper, SHVFY)) != NOSTRFOUND) &&
        ((i = sindex(oper, ATTR))  != NOSTRFOUND)  )

    {
        (void) strcpy(oper, UNKWN);
    }
}

/* If the user had multiple name-value pairs on   */
/* this line, you will have to check if an error  */
/* or the operation was completed as single line  */
/* input.                                         */
/*                                                */
/* form=9v13&out,idpname="idp-2,incdigits=1",out! */

if ((i = sindex(tbuf, ",")) != NOSTRFOUND)
{
    /* Multiple name-value pairs for this input   */
    /* line. Remove the "Form=" part from the     */
    /* temporary buffer and re-parse the current  */
    /* line.                                      */

    tbuf[pos]   = '?';   /* Replace 'f' */
```

```
                    /* Now the search for the "Form=" string will */
                    /* fail on the next pass.                     */

                    goto parse_again;
                }

            }   /* end else Cur_viewnum < MAXVIEW */

        }   /* else Cur_classnum < MAXCLASS */
    }

    else if (foundForm == FALSE)
    {

        /***********************/
        /* WAITING FOR FORM=   */
        /***********************/

        /* Ignore what you got */

        continue;
    }

    else if ( ((pos = sindex(tbuf, EXPASS)) != NOSTRFOUND) ||
              ((pos = sindex(tbuf, EXFAIL)) != NOSTRFOUND) )
    {
        /************************************************/
        /* Application: Expecting a PASS when it FAILED */
        /*                                              */
        /*              - OR -                          */
        /*                                              */
        /* Application: Expecting a FAIL when it PASSED */
        /************************************************/

        /* Set the error mode to what according to what the    */
        /* what the application expected not per the '#screen' */
        /* command because the '#PASS' and '#FAIL' lines       */
        /* are not in the output unless you execute the        */
        /* '-e' command line argument for the application.     */
```

```
            /*                                                      */
            /*  'Application: a {PASS | FAIL} was expected'         */

            if ((pos = sindex(tbuf, EXPASS)) != NOSTRFOUND)
                (void) strcpy(emode, SPASS);
            else
                (void) strcpy(emode, SFAIL);

            (void) strcpy(ename, sname);
            (void) sprintf(elastscreen, "%s\n", lastscreen);

            rc_err     = TRUE;
            rc_done    = TRUE;
            foundForm  = FALSE;
        }

        else if ((pos = sindex(tbuf, oper)) != NOSTRFOUND)
        {
            /***********************************/
            /* CURRENT transation HAS FINISHED */
            /***********************************/

            rc_done = TRUE;

            (void) strcpy(ename, sname);
            (void) sprintf(elastscreen, "%s\n", lastscreen);

            /* set the lines read since the rc completed to 0 */

            done_read = 0;

        }

        else if ( ((pos = sindex(tbuf, STOP)) != NOSTRFOUND) ||
                  ((pos = sindex(tbuf, EXIT)) != NOSTRFOUND)  ||
                  ((pos = sindex(tbuf, PF)) != NOSTRFOUND))
        {
                /********************/
                /* STOP, EXIT or PF */
```

```
903                 /********************/
904
905                 foundForm = FALSE;
906             }
907
908             /*********************************************************/
909             /* Allow 15 lines after completing an operation to send  */
910             /* "Application expecting..."                            */
911             /*********************************************************/
912
913             if (rc_done == TRUE)
914                 done_read++;
915
916             if (done_read >= 15)
917             {
918                 foundForm = FALSE;
919                 done_read = 0;
920
921                 continue;
922             }
923
924             /*****************************************************/
925             /* ADD 1 TO linenum AND GET NEXT APPTEXT OUTPUT LINE */
926             /*****************************************************/
927
928
929             lineno++;
930             if (lineno >= MAXLINENO)
931             {
932                 (void) fprintf(stderr,
933                                "\nError:  Input data exceeds maximum array size\n");
934                 (void) fprintf(stderr, "BUFFER CONTAINS -\n");
935
936                 for (k=0; k<=lineno; k++)
937                     (void) fprintf(stderr,
938                                    "Inbuf[%d]: %s", k, Inbuf[k]);
939                 (void) fprintf(stderr, "\n");
940                 exit(1);
941
```

```
942             }
943
944         }   /* end of while (TRUE) */
945
946
947         /*****************************************/
948         /* UPDATE THE STATS FROM THE PREVIOUS RC */
949         /*****************************************/
950
951
952         /* Let A views occupy 0th element of array.           */
953         /* Let B views occupy 1st element of array.           */
954         /* Let other views occupy +2 offset element of array. */
955
956         uclass = Cur_classnum;
957         if (uclass == 0)
958         {
959             /* uclass is zero when Cur_classnum encountered a */
960             /* non-numeric number -- classes A and B          */
961
962             if (Cur_class[0] == 'b')
963                 uclass++;
964         }
965         else
966         {
967             uclass = uclass + 2;
968             if (uclass >= MAXCLASS)
969                 uclass = MAXCLASS - 1;
970         }
971         updstat(rc_err, uclass, Cur_viewnum, lineno, oper, sname);
972
973         if (rc_err == TRUE)
974         {
975             for (i=0; i<lineno; i++)
976             {
977                if ((pos = sindex(Inbuf[i], MSGU)) != NOSTRFOUND)
978                    break;
979             }
980
```

```
981             if (i == lineno)
982                {
983                 /* Look for 1st a FAIL was expected        */
984                 /* Expected fails will have this line      */
985
986                 for (i=0; i<lineno; i++)
987                 {
988                    if ((pos = sindex(Inbuf[i], MEXFAIL)) != NOSTRFOUND)
989                        break;
990                 }
991                 if (i == lineno)
992                 {
993
994                     (void) fprintf(stderr,
995                            "\nError:  Can't find [%s or %s] error line\n",
996                            MSGU, MEXFAIL);
997
998                     (void) fprintf(stderr, "BUFFER CONTAINS -\n");
999
1000                    for (j=0; j<=lineno; j++)
1001                        (void) fprintf(stderr,
1002                                       "Inbuf[%d]: %s", j, Inbuf[j]);
1003
1004                    (void) fprintf(stderr, "\n");
1005
1006                       exit(1);
1007                }
1008                else
1009                {
1010                    errstart = errstop = i - 1;
1011                }
1012               }
1013            else
1014               {
1015                errstart = i;
1016
1017                /*********************************/
1018                /* FIND ?E no more error messages */
1019                /*********************************/
```

```
020
021                 for (j=i; j<lineno; j++)
022                 {
023                    if ((pos = sindex(Inbuf[j], MNOMORE)) != NOSTRFOUND)
024                        break;
025                 }
026
027                 if (j == lineno)
028                    {
029                     (void) fprintf(stderr,
030                                  "\nError:  Can't find [%s] error line\n",
031                                  MNOMORE);
032
033                     (void) fprintf(stderr, "BUFFER CONTAINS -\n");
034
035                     for (k=0; k<=lineno; k++)
036                         (void) fprintf(stderr,
037                                "Inbuf[%d]: %s", k, Inbuf[k]);
038                     (void) fprintf(stderr, "\n");
039
040                        exit(1);
041                    }
042
043                  errstop = j - 1;
044                }
045
046              /*****************************/
047              /* PROCESS THE ERROR MESSAGE */
048              /*****************************/
049
050                 prcs_err(lineno-1, emode, uclass, Cur_viewnum, ename,
051                          tname, inpkg, elastscreen, errstart, errstop);
052
053          }
054
055       }   /* end of for (arg_no = optind to argc) */
056
057       /* Set the error flag if errors occured */
058
```

```
059         if (Tstfail > 0)
060            Errflag = 1;
061         else
062            Errflag = 0;
063
064         (void) fprintf(stdout, "\n\n* * *   E N D   O F   J O B   * * *\n\n");
065         (void) fprintf(stdout, "ANALYSIS COMPLETE FOR: %s \n", inpkg);
066
067         /****************************/
068         /* PRINT THE JOB STATISTICS */
069         /****************************/
070
071         Tottests  = Tstpass + Tstfail;
072         passper   = (float) (100 * Tstpass) / (float) Tottests;
073
074
075         /* Perform a page eject for the Output file report */
076
077         (void) fprintf(Ofp, "%c\n", '
```

```
078 ');
079
080       if (caption[0] != '\0')
081       {
082           (void) sprintf(Inbuf[0], "Application %s Analysis", caption);
083       }
084       else
085       {
086           (void) strcpy(Inbuf[0], "Application Analysis");
087       }
088
089       capcntr(RPTWIDTH, Inbuf[0]);
090       (void) strcat(Inbuf[0], "\n");
091
092          (void) sprintf(Inbuf[1], "%s", time_buf);
093       capcntr(RPTWIDTH, Inbuf[1]);
094       (void) strcat(Inbuf[1], "\n\n");
095
096       if (su_package == TRUE)
097       {
098           (void) sprintf(Inbuf[2], "%s %s %s %s",
099                                    HDRSTR0, usrgen, HDRSTR1, pack_name);
100       }
101       else
102       {
103           (void) sprintf(Inbuf[2], "%s %s %s %s",
104                                    HDRSTR0, usrgen, HDRSTR3, pack_name);
105       }
106       capcntr(RPTWIDTH, Inbuf[2]);
107       (void) strcat(Inbuf[2], "\n");
108
109       (void) sprintf(Inbuf[3],
110            "\n                    Total Input Lines Read      = %10ld\n\n",
111                                                          Tlines);
112       (void) sprintf(Inbuf[4],
113            "                     SUCCESSFUL RC Operations\n");
114
115       (void) sprintf(Inbuf[5],
116            "                           + NEW     Operations  = %10ld\n",
```

```
                                                        Pnew);

      (void) sprintf(Inbuf[6],
           "                          + CHG     Operations  = %10ld\n",
                                                        Pchg);

      (void) sprintf(Inbuf[7],
           "                          + OUT     Operations  = %10ld\n",
                                                        Pout);

      (void) sprintf(Inbuf[8],
           "                          + VFY     Operations  = %10ld\n",
                                                        Pvfy);
      (void) sprintf(Inbuf[9],
           "                          + MVFY    Operations  = %10ld\n",
                                                        Pmvfy);

      (void) sprintf(Inbuf[10],
           "                          + SHVFY   Operations  = %10ld\n",
                                                        Pshvfy);

      (void) sprintf(Inbuf[11],
           "                          + ATTR    Operations  = %10ld\n",
                                                        Pattr);
      (void) sprintf(Inbuf[12],
           "                          + UNKNOWN Operations  = %10ld\n",
                                                       Punkwn);

      (void) sprintf(Inbuf[13],
            "                                                ----------\n");

      (void) sprintf(Inbuf[14],
           "                     Total Tests PASSED         = %10ld\n\n",
                                                        Tstpass);
      (void) sprintf(Inbuf[15],
           "                     FAILED Operations\n");
      (void) sprintf(Inbuf[16],
           "                          + NEW     Operations  = %10ld\n",
                                                        Fnew);
```

```
156
157         (void) sprintf(Inbuf[17],
158              "                          + CHG     Operations  = %10ld\n",
159                                           Fchg);
160
161         (void) sprintf(Inbuf[18],
162              "                          + OUT     Operations  = %10ld\n",
163                                           Fout);
164
165         (void) sprintf(Inbuf[19],
166              "                          + VFY     Operations  = %10ld\n",
167                                           Fvfy);
168         (void) sprintf(Inbuf[20],
169              "                          + MVFY    Operations  = %10ld\n",
170                                           Fmvfy);
171
172         (void) sprintf(Inbuf[21],
173              "                          + SHVFY   Operations  = %10ld\n",
174                                           Fshvfy);
175
176         (void) sprintf(Inbuf[22],
177              "                          + ATTR    Operations  = %10ld\n",
178                                           Fattr);
179         (void) sprintf(Inbuf[23],
180              "                          + UNKNOWN Operations  = %10ld\n",
181                                           Funkwn);
182
183         (void) sprintf(Inbuf[24],
184                "                                                ----------\n");
185
186         (void) sprintf(Inbuf[25],
187                "                    Total Tests   FAILED        = %10ld\n\n",
188                                           Tstfail);
189         (void) sprintf(Inbuf[26],
190                "                    Total Unique Error Messages = %10d\n\n",
191                                           Nperr.entries);
192         (void) sprintf(Inbuf[27],
193                "                    Total Tests                 = %10ld\n\n",
194                                           Tottests);
```

```
195         (void) sprintf(Inbuf[28],
196                "                    Total Scripts FAILED       = %10d\n\n",
197                                                                 Serr_cnt);
198         (void) sprintf(Inbuf[29],
199                "                    Success PASS Ratio         =   %8.2f %%\n\n",
200                                                                 passper);
201         /*******************************/
202         /* DO WE HAVE RUN HISTORY DATA? */
203         /*******************************/
204
205         if ((Runhist.start[0] != '\0') && (Runhist.end[0] != '\0'))
206         {
207              /* we calculated it */
208
209             j = 35;
210
211             (void) sprintf(Inbuf[30], "\n\nRUN DURATION:\n\n");
212
213             (void) sprintf(Inbuf[31],
214                            "     Start Day: %s     Start Time: %s\n",
215                                Runhist.start_day, Runhist.start_time);
216
217             (void) sprintf(Inbuf[32],
218                            "     End   Day: %s     End   Time: %s\n\n",
219                                Runhist.end_day, Runhist.end_time);
220
221             (void) sprintf(Inbuf[33], "ELAPSED TIME:\n\n");
222
223             (void) sprintf(Inbuf[34],
224                            "     %2d Days  %2d Hours  %2d Minutes  %2d Seconds\n",
225                         Runhist.days, Runhist.hours, Runhist.mins,
226                         Runhist.secs);
227
228
229         }
230         else
231         {
232              /* we DIDN'T calculated it */
233
```

```
234             j = 30;
235         }
236
237         for (i=0; i<j; i++)
238         {
239             (void) fprintf(Rfp, "%s", Inbuf[i]);
240             (void) fprintf(Ofp, "%s", Inbuf[i]);
241         }
242
243
244         /****************************************/
245         /* DISPLAY THE SUCCESSFUL TEST COVERAGE */
246         /****************************************/
247
248         lineno = 0;
249            pageno = 0;
250
251         /* Build header line Inbuf[0] */
252
253         if (caption[0] != '\0')
254         {
255             (void) sprintf(thdr0,
256                            "SUCCESSFUL Application %s Test Coverage",
257                             caption);
258         }
259         else
260         {
261             (void) sprintf(thdr0,
262                            "SUCCESSFUL Application Test Coverage");
263         }
264
265         /* Build header line Inbuf[3] with a line of equal signs */
266
267         i = strlen(thdr0);
268         for (j=0; j<i; j++)
269             thdr3[j] = '=';
270         thdr3[j] = '\0';
271
272         /* center header line 1 */
```

```
273
274         capcntr(RPTWIDTH, thdr0);
275
276         /* center header Inbuf[3] */
277
278         capcntr(RPTWIDTH, thdr3);
279         (void) sprintf(Inbuf[3], "%s\n\n", thdr3);
280
281         /* Prepare header line 1 to contain the Page % literal 8 columns      */
282         /* from the end of the page width by filling with blanks to that point*/
283
284         i = strlen(thdr0);
285
286         for (j=i; j < RPTWIDTH-8; j++)
287             thdr0[j] = ' ';
288         thdr0[j] = '\0';
289
290         for (i=0; i<MAXCLASS; i++)
291         {
292             for(j=0; j<MAXVIEW; j++)
293                {
294                 if (Class[i].View[j].Pass.count > 0)
295                 {
296                     if ((pageno == 0) || (lineno > 59))
297                     {
298                         pageno++;
299                         (void) sprintf(Inbuf[0], "
```

```
\n%s Page %d\n",
                                    thdr0, pageno);

                    /* Use Inbuf[1] and Inbuf[2] from previous page */

                    (void) sprintf(Inbuf[4],
                    "      VIEW           ................... OPERATION TYPES
...................\n");

                    (void) sprintf(Inbuf[5],
                    "     NUMBER  TOTAL    NEW    CHG    OUT    VFY   MVFY   SHVFY  ATTR
UNKWN\n");

                    (void) sprintf(Inbuf[6],
                    "     ====== ======  ====== ====== ====== ====== ====== ====== ======
======\n\n");

                    for (k=0; k<7; k++)
                    {
                        (void) fprintf(Rfp, "%s", Inbuf[k]);
                        (void) fprintf(Ofp, "%s", Inbuf[k]);
                    }
                    lineno = 11;
                }

                /* Remember that the array has the A & and B views in     */
                /* array 0 and 1 elements.                                */

                if (i < 2)
                {
                    if (i == 0)
                       (void) sprintf(tbuf, " A.%-2d",j);
                    else
                       (void) sprintf(tbuf, " B.%-2d",j);
                }
                else
                {
                    if (i == MAXCLASS - 1)
                        (void) sprintf(tbuf, "UNKWN");
```

```
                    else
                        (void) sprintf(tbuf, "%2d.%-2d", i-2, j);
                }


                (void) sprintf(Inbuf[0],
                "     %s  %6d  %6d %6d %6d %6d %6d %6d %6d %6d\n",
                                tbuf,
                                Class[i].View[j].Pass.count,
                                Class[i].View[j].Pass.new,
                                Class[i].View[j].Pass.chg,
                                Class[i].View[j].Pass.out,
                                Class[i].View[j].Pass.vfy,
                                Class[i].View[j].Pass.mvfy,
                                Class[i].View[j].Pass.shvfy,
                                Class[i].View[j].Pass.attr,
                                Class[i].View[j].Pass.unkwn);
                lineno++;

                (void) fprintf(Rfp, "%s", Inbuf[0]);
                (void) fprintf(Ofp, "%s", Inbuf[0]);

             }   /* end if Class[i].View[j].Pass.count > 0 */

           }   /* end for j=0 to MAXVIEW */

    }   /* end for i=0 to MAXCLASS */

    /****************************/
    /* DISPLAY THE FAILED TESTS */
    /****************************/

    if (Errflag == 0)
        goto disp_res;

    lineno = 0;
       pageno = 0;

    /* Build header line Inbuf[0] */
```

```
378
379         if (caption[0] != '\0')
380         {
381             (void) sprintf(thdr0,
382                            "FAILED Application %s Test Coverage",
383                             caption);
384         }
385         else
386         {
387             (void) sprintf(thdr0,
388                            "FAILED Application Test Coverage");
389         }
390
391         /* Build header line Inbuf[3] with a line of equal signs */
392
393         i = strlen(thdr0);
394         for (j=0; j<i; j++)
395             thdr3[j] = '=';
396         thdr3[j] = '\0';
397
398         /* center header Inbuf[0] */
399
400         capcntr(RPTWIDTH, thdr0);
401
402         /* center header Inbuf[3] */
403
404         capcntr(RPTWIDTH, thdr3);
405         (void) sprintf(Inbuf[3], "%s\n\n", thdr3);
406
407         /* Prepare header line 1 to contain the Page % literal 8 columns      */
408         /* from the end of the page width by filling with blanks to that point*/
409
410         i = strlen(thdr0);
411
412         for (j=i; j < RPTWIDTH-8; j++)
413             thdr0[j] = ' ';
414         thdr0[j] = '\0';
415
416         for (i=0; i<MAXCLASS; i++)
```

```
417         {
418             for(j=0; j<MAXVIEW; j++)
419                {
420                 if (Class[i].View[j].Fail.count > 0)
421                 {
422                     if ((pageno == 0) || (lineno > 59))
423                     {
424                         pageno++;
425                         (void) sprintf(Inbuf[0], "
```

```
\n%s Page %d\n",
                                    thdr0, pageno);

                    /* Use Inbuf[1] and Inbuf[2] from previous page */

                    (void) sprintf(Inbuf[4],
                    "      VIEW           ................... OPERATION TYPES
...................\n");

                    (void) sprintf(Inbuf[5],
                    "     NUMBER  TOTAL    NEW    CHG    OUT    VFY   MVFY   SHVFY  ATTR
UNKWN\n");

                    (void) sprintf(Inbuf[6],
                    "     ====== ======  ====== ====== ====== ====== ====== ====== ======
======\n\n");

                    for (k=0; k<7; k++)
                    {
                        (void) fprintf(Rfp, "%s", Inbuf[k]);
                        (void) fprintf(Ofp, "%s", Inbuf[k]);
                    }
                    lineno = 11;
                }

                /* Remember that the array has the A & and B views in     */
                /* array 0 and 1 elements.                                */

                if (i < 2)
                {
                    if (i == 0)
                       (void) sprintf(tbuf, " A.%-2d",j);
                    else
                       (void) sprintf(tbuf, " B.%-2d",j);
                }
                else
                {
                    if (i == MAXCLASS - 1)
                        (void) sprintf(tbuf, "UNKWN");
```

```
                    else
                        (void) sprintf(tbuf, "%2d.%-2d", i-2, j);
                }


                (void) sprintf(Inbuf[0],
                "     %s  %6d  %6d %6d %6d %6d %6d %6d %6d %6d\n",
                                tbuf,
                                Class[i].View[j].Fail.count,
                                Class[i].View[j].Fail.new,
                                Class[i].View[j].Fail.chg,
                                Class[i].View[j].Fail.out,
                                Class[i].View[j].Fail.vfy,
                                Class[i].View[j].Fail.mvfy,
                                Class[i].View[j].Fail.shvfy,
                                Class[i].View[j].Fail.attr,
                                Class[i].View[j].Fail.unkwn);
                lineno++;

                (void) fprintf(Rfp, "%s", Inbuf[0]);
                (void) fprintf(Ofp, "%s", Inbuf[0]);

             }   /* end if Class[j].View[j].Fail.count > 0 */

            }   /* end for j=0 to MAXVIEW */

     }   /* end for i=0 to MAXCLASS */


              /************************************************/
disp_res:     /* DISPLAY THE Application VIEW ANALYSIS RESULTS */
              /************************************************/

     lineno = 0;
        pageno = 0;

     /* Build header line Inbuf[0] */

     if (caption[0] != '\0')
```

```
504         {
505             (void) sprintf(thdr0,
506                            "Application %s View Analysis",
507                             caption);
508         }
509         else
510         {
511             (void) sprintf(thdr0,
512                            "Application %s View Analysis");
513         }
514
515         /* Build header line Inbuf[3] with a line of equal signs */
516
517         i = strlen(thdr0);
518         for (j=0; j<i; j++)
519             thdr3[j] = '=';
520         thdr3[j] = '\0';
521
522         /* center header Inbuf[0] */
523
524         capcntr(RPTWIDTH, thdr0);
525
526         /* center header Inbuf[3] */
527
528         capcntr(RPTWIDTH, thdr3);
529         (void) sprintf(Inbuf[3], "%s\n\n", thdr3);
530
531         /* Prepare header line 1 to contain the Page % literal 8 columns      */
532         /* from the end of the page width by filling with blanks to that point*/
533
534         i = strlen(thdr0);
535
536         for (j=i; j < RPTWIDTH-8; j++)
537             thdr0[j] = ' ';
538         thdr0[j] = '\0';
539
540         for (i=0; i<MAXCLASS; i++)
541         {
542             for(j=0; j<MAXVIEW; j++)
```

```
{
 if ( (Class[i].View[j].Fail.count > 0) ||
      (Class[i].View[j].Pass.count > 0) )
 {
     if ((pageno == 0) || (lineno > 59))
     {
         pageno++;
         (void) sprintf(Inbuf[0], "
```

```
\n%s Page %d\n",
                                thdr0, pageno);

                    /* Use Inbuf[1] and Inbuf[2] from previous page */

                    (void) sprintf(Inbuf[4],
              "     VIEW      OPERATION     PASSED     FAILED      TOTAL     PASS      FAIL \n");

                    (void) sprintf(Inbuf[5],
              "    NUMBER       TYPE        TESTS      TESTS       TESTS     RATIO     RATIO\n");

                    (void) sprintf(Inbuf[6],
              "    ======     =========     ======     ======      =====     =====     =====\n");

                    for (k=0; k<7; k++)
                    {
                        (void) fprintf(Rfp, "%s", Inbuf[k]);
                        (void) fprintf(Ofp, "%s", Inbuf[k]);
                    }
                    /* Set line number equal to 9 lines of heading     */
                    /*                         + 7 lines of detail     */

                    lineno = 16;
                }

                    /**************************/
                    /* Capture the view number */
                    /**************************/

                /* Remember that the array has the A & and B views in     */
                /* array 0 and 1 elements.                                */

                if (i < 2)
                {
                    if (i == 0)
                       (void) sprintf(tbuf, " A.%-2d",j);
                    else
                       (void) sprintf(tbuf, " B.%-2d",j);
                }
```

```
590                 else
591                 {
592                     if (i == MAXCLASS - 1)
593                         (void) sprintf(tbuf, "UNKWN");
594                     else
595                         (void) sprintf(tbuf, "%2d.%-2d", i-2, j);
596                 }
597
598
599                 /*********************************/
600                 /* Display the NEW operation type */
601                 /*********************************/
602
603                 Tstpass = Class[i].View[j].Pass.new;
604                 Tstfail = Class[i].View[j].Fail.new;
605
606                 Tottests = Tstpass + Tstfail;
607
608                 if (Tstpass == 0L)
609                    passper = 0.0;
610                 else
611                    passper = (float) (100 * Tstpass) / (float) Tottests;
612
613                 if (Tstfail == 0L)
614                    failper = 0.0;
615                 else
616                    failper = (float) (100 * Tstfail) / (float) Tottests;
617
618                 (void) sprintf(Inbuf[0],
619                 "\n    %s         NEW        %6d     %6d     %6d     %5.1f     %5.1f\n",
620                                   tbuf,
621                               Tstpass,
622                               Tstfail,
623                               Tottests,
624                               passper,
625                               failper);
626
627                 (void) fprintf(Rfp, "%s", Inbuf[0]);
628                 (void) fprintf(Ofp, "%s", Inbuf[0]);
```

```
629
630                 /*********************************/
631                 /* Display the CHG operation type */
632                 /*********************************/
633
634                 Tstpass = Class[i].View[j].Pass.chg;
635                 Tstfail = Class[i].View[j].Fail.chg;
636
637                 Tottests = Tstpass + Tstfail;
638
639                 if (Tstpass == 0L)
640                    passper = 0.0;
641                 else
642                    passper = (float) (100 * Tstpass) / (float) Tottests;
643
644                 if (Tstfail == 0L)
645                    failper = 0.0;
646                 else
647                    failper = (float) (100 * Tstfail) / (float) Tottests;
648
649                 (void) sprintf(Inbuf[0],
650                 "                  CHG        %6d     %6d     %6d     %5.1f     %5.1f\n",
651                               Tstpass,
652                               Tstfail,
653                               Tottests,
654                               passper,
655                               failper);
656
657                 (void) fprintf(Rfp, "%s", Inbuf[0]);
658                 (void) fprintf(Ofp, "%s", Inbuf[0]);
659
660                 /*********************************/
661                 /* Display the OUT operation type */
662                 /*********************************/
663
664                 Tstpass = Class[i].View[j].Pass.out;
665                 Tstfail = Class[i].View[j].Fail.out;
666
667                 Tottests = Tstpass + Tstfail;
```

```
668
669                 if (Tstpass == 0L)
670                    passper = 0.0;
671                 else
672                    passper = (float) (100 * Tstpass) / (float) Tottests;
673
674                 if (Tstfail == 0L)
675                    failper = 0.0;
676                 else
677                    failper = (float) (100 * Tstfail) / (float) Tottests;
678
679                 (void) sprintf(Inbuf[0],
680                 "                  OUT        %6d     %6d     %6d     %5.1f     %5.1f\n",
681                               Tstpass,
682                               Tstfail,
683                               Tottests,
684                               passper,
685                               failper);
686
687                 (void) fprintf(Rfp, "%s", Inbuf[0]);
688                 (void) fprintf(Ofp, "%s", Inbuf[0]);
689
690
691                 /*********************************/
692                 /* Display the VFY operation type */
693                 /*********************************/
694
695                 Tstpass = Class[i].View[j].Pass.vfy
696                         + Class[i].View[j].Pass.mvfy
697                         + Class[i].View[j].Pass.shvfy
698                         + Class[i].View[j].Pass.attr ;
699
700                 Tstfail = Class[i].View[j].Fail.vfy
701                         + Class[i].View[j].Fail.mvfy
702                         + Class[i].View[j].Fail.shvfy
703                         + Class[i].View[j].Fail.attr ;
704
705                 Tottests = Tstpass + Tstfail;
706
```

```
707                  if (Tstpass == 0L)
708                     passper = 0.0;
709                  else
710                     passper = (float) (100 * Tstpass) / (float) Tottests;
711
712                  if (Tstfail == 0L)
713                     failper = 0.0;
714                  else
715                     failper = (float) (100 * Tstfail) / (float) Tottests;
716
717                  (void) sprintf(Inbuf[0],
718                  "                  VFY        %6d     %6d     %6d     %5.1f     %5.1f\n",
719                                Tstpass,
720                                Tstfail,
721                                Tottests,
722                                passper,
723                                failper);
724
725                  (void) fprintf(Rfp, "%s", Inbuf[0]);
726                  (void) fprintf(Ofp, "%s", Inbuf[0]);
727
728                  /*******************************************/
729                  /* Display the TOTAL for this operation type */
730                  /*******************************************/
731
732                  Tstpass = Class[i].View[j].Pass.count;
733                  Tstfail = Class[i].View[j].Fail.count;
734
735                  Tottests = Tstpass + Tstfail;
736                  if (Tstpass == 0L)
737                     passper = 0.0;
738                  else
739                     passper = (float) (100 * Tstpass) / (float) Tottests;
740
741                  if (Tstfail == 0L)
742                     failper = 0.0;
743                  else
744                     failper = (float) (100 * Tstfail) / (float) Tottests;
745
```

```
746                 (void) sprintf(Inbuf[0],
747                 "                 -----        ------     ------     ------     -----     -----
748 \n");
749
750                 (void) fprintf(Rfp, "%s", Inbuf[0]);
751                 (void) fprintf(Ofp, "%s", Inbuf[0]);
752
753                 (void) sprintf(Inbuf[0],
754                 "                TOTAL        %6d     %6d     %6d     %5.1f     %5.1f\n",
755                                 Tstpass,
756                                 Tstfail,
757                                 Tottests,
758                                 passper,
759                                 failper);
760
761                 (void) fprintf(Rfp, "%s", Inbuf[0]);
762                 (void) fprintf(Ofp, "%s", Inbuf[0]);
763
764                 lineno = lineno + 7;
765
766              }   /* end if Class[j].View[j].Pass.count > 0 */
767                  /* AND    Class[j].View[j].Fail.count > 0 */
768
769             }   /* end for j=0 to MAXVIEW */
770
771       }   /* end for i=0 to MAXCLASS */
772
773       if (Errflag == 0)
774           goto end_run;
775
776
777       /*********************************************************/
778       /* SORT THE APPLICATION NAME ARRAY USING A HEAP SORT METHOD */
779       /*********************************************************/
780
781       /* Determine the number of entries to create the heap (pos) */
782
783
784       if (Serr_cnt < 2)
```

```
785         {
786             /* Nothing to sort -- just display what you got */
787
788             (void) fprintf(stderr,
789                         "\nWARNING:  Script Names are NOT sorted\n");
790
791             goto show_names;
792         }
793
794         /* Create the heap */
795
796         crheap(Serr_cnt);
797
798         /* Rc structure is now in heap order --- now sort it */
799
800         for (last_mbr=Serr_cnt; last_mbr>1; last_mbr--)
801         {
802             /* Exchange the last member and ther first member */
803             /* because the first member is the highest member */
804             /* for the current heap.                          */
805
806             (void) strcpy(temp_name, Rc[1].name);
807             temp_count = Rc[1].count;
808
809             (void) strcpy(Rc[1].name, Rc[last_mbr].name);
810             Rc[1].count = Rc[last_mbr].count;
811
812             (void) strcpy(Rc[last_mbr].name, temp_name);
813             Rc[last_mbr].count = temp_count;
814
815
816             /* Initialize the data */
817
818             parent    = 1;
819             great_son = 2;
820             key_count = Rc[1].count;
821             (void) strcpy(key_name, Rc[1].name);
822
823             /* Find the greatest son */
```

```
824
825         if ((great_son + 1) < last_mbr)
826             if ( (strcmp(Rc[great_son + 1].name, Rc[great_son].name))
827                         > 0 )
828                 great_son++;
829
830         /************************/
831         /* Reconstruct new heap */
832         /************************/
833
834         while ( (great_son <= last_mbr - 1) &&
835                 (strcmp(Rc[great_son].name, key_name) > 0) )
836         {
837             /* Exchange Parent and Son */
838
839             (void) strcpy(Rc[parent].name, Rc[great_son].name);
840             Rc[parent].count = Rc[great_son].count;
841
842             /* Obtain the next left son */
843
844             parent    = great_son;
845             great_son = 2 * parent;
846
847             /* Find the greatest son */
848
849             if (great_son + 1 < last_mbr)
850             {
851                 if ((strcmp(Rc[great_son + 1].name, Rc[great_son].name))
852                             > 0)
853                     great_son++;
854             }
855             else
856             {
857                 /* Check the boundary of the array */
858
859                 if (great_son > Serr_cnt)
860                     great_son = Serr_cnt;
861             }
862
```

```
863             /* Make the Parent the key */
864
865             (void) strcpy(Rc[parent].name, key_name);
866             Rc[parent].count = key_count;
867
868         }   /* end while -- reconstruct new heap */
869
870     }   /* end of for */
871
872 show_names:
873
874     /***********************************/
875     /* DISPLAY THE FAILED SCRIPT NAMES */
876     /***********************************/
877
878        pageno = 0;
879        lineno = 1;
880
881     /* Build header line Inbuf[0] */
882
883     if (caption[0] != '\0')
884     {
885         (void) sprintf(thdr0,
886                        "Application %s Script Failures",
887                         caption);
888     }
889     else
890     {
891         (void) sprintf(thdr0,
892                        "Application Script Failures");
893     }
894
895     /* Build header line Inbuf[3] with a line of equal signs */
896
897     i = strlen(thdr0);
898     for (j=0; j<i; j++)
899         thdr3[j] = '=';
900     thdr3[j] = '\0';
901
```

```
902         /* center header Inbuf[0] */
903
904         capcntr(RPTWIDTH, thdr0);
905
906         /* center header Inbuf[3] */
907
908         capcntr(RPTWIDTH, thdr3);
909         (void) sprintf(Inbuf[3], "%s\n\n", thdr3);
910
911         /* Prepare header line 1 to contain the Page % literal 8 columns      */
912         /* from the end of the page width by filling with blanks to that point*/
913
914         i = strlen(thdr0);
915
916         for (j=i; j < RPTWIDTH-8; j++)
917             thdr0[j] = ' ';
918         thdr0[j] = '\0';
919
920         while (TRUE)
921         {
922             pageno++;
923             (void) sprintf(Inbuf[0], "
```

```
924   \n%s Page %d\n", thdr0, pageno);
925
926           /* Use Inbuf[1] and Inbuf[2] from previous page */
927
928           (void) sprintf(Inbuf[4],
929           "   RC SCRIPT NAME        ERROR COUNT        RC SCRIPT NAME        ERROR COUNT\n");
930
931           (void) sprintf(Inbuf[5],
932           "====================     ===========     ====================     ===========\n");
933
934           for (i=0; i<6; i++)
935              {
936               (void) fprintf(Rfp, "%s", Inbuf[i]);
937               (void) fprintf(Ofp, "%s", Inbuf[i]);
938              }
939
940           /*************************************************************/
941           /* Calculate how many lines to print in column1 and column 2 */
942           /*************************************************************/
943
944           if ((lineno + PRTMAX1) > Serr_cnt)
945           {
946              to_prt = Serr_cnt - lineno + 1;
947           }
948           else
949           {
950              to_prt = PRTMAX1;
951           }
952           pos = to_prt/2;
953
954           /* Have the odd numbers print 1 more entry in column 1 */
955
956           if ((pos * 2) < to_prt)
957               pos1end = pos + 1;
958           else
959               pos1end = pos;
960
961           for (i=lineno; i<lineno+pos1end; i++)
962           {
```

```
963             /* Verify that column 2 really exists */
964
965             if ((i + pos1end) > Serr_cnt)
966             {
967                 (void) sprintf(Inbuf[0],
968                                "%-20s         %5d\n",
969                                Rc[i].name, Rc[i].count);
970
971                 /* Use j to tell you the last line printed */
972                 j = i;
973             }
974             else
975             {
976                 (void) sprintf(Inbuf[0],
977                               "%-20s         %5d       %-20s         %5d\n",
978                               Rc[i].name, Rc[i].count,
979                               Rc[i+pos1end].name, Rc[i+pos1end].count);
980
981                 /* Use j to tell you the last line printed */
982                 j = i + pos1end;
983             }
984
985             (void) fprintf(Rfp, "%s", Inbuf[0]);
986             (void) fprintf(Ofp, "%s", Inbuf[0]);
987
988         }
989         if (to_prt != PRTMAX1)
990             break;
991
992         lineno = lineno + PRTMAX1;
993
994         if (j >= Serr_cnt)
995             break;
996     }
997
998     /*****************************************/
999     /* DISPLAY THE UNIQUE ERROR MESSAGE LISTING */
000     /*****************************************/
001
```

```
002         lineno = 0;
003            pageno = 0;
004
005         /* Build header line Inbuf[0] */
006
007         if (caption[0] != '\0')
008         {
009             (void) sprintf(thdr0,
010                           "UNIQUE Application %s Errror Message Listing",
011                            caption);
012         }
013         else
014         {
015             (void) sprintf(thdr0,
016                           "UNIQUE Application s Errror Message Listing");
017         }
018
019         /* Build header line Inbuf[3] with a line of equal signs */
020
021         i = strlen(thdr0);
022         for (j=0; j<i; j++)
023             thdr3[j] = '=';
024         thdr3[j] = '\0';
025
026         /* center header line 1 */
027
028         capcntr(RPTWIDTH, thdr0);
029
030         /* center header line 3 */
031
032         capcntr(RPTWIDTH, thdr3);
033         (void) sprintf(Inbuf[3], "%s\n\n", thdr3);
034
035         /* Prepare header line 1 to contain the Page % literal 8 columns      */
036         /* from the end of the page width by filling with blanks to that point*/
037
038         i = strlen(thdr0);
039
040         for (j=i; j < RPTWIDTH-8; j++)
```

```
041             thdr0[j] = ' ';
042         thdr0[j] = '\0';
043
044         for (i=1; i <= Nperr.entries; i++)
045         {
046             /* Calculate the number of error message lines to print */
047
048             lineno += Nperr.nerr[i].no_lines + 2;
049
050             if ((pageno == 0) || (lineno >= 59))
051             {
052                 pageno++;
053                 (void) sprintf(Inbuf[0], "
```

```
\n%s Page %d\n",
                            thdr0, pageno);

            /* Use Inbuf[1] and Inbuf[2] from previous page */


            for (j=0; j<4; j++)
            {
                (void) fprintf(Rfp, "%s", Inbuf[j]);
                (void) fprintf(Ofp, "%s", Inbuf[j]);
            }
            lineno = 6 ;
            lineno += Nperr.nerr[i].no_lines + 2;
        }

        (void) fprintf(Rfp, "%5d. MESSAGE OCCURRED: %5d
.............................................\n",
                        i, Nperr.nerr[i].no_entries);
        (void) fprintf(Ofp, "%5d. MESSAGE OCCURRED: %5d
.............................................\n",
                        i, Nperr.nerr[i].no_entries);

        /* Loop through the errindx to pull out each error message out of */
        /* the error dictionary.  A space will seperate each index.       */

        indxlen    = strlen(Nperr.nerr[i].errindx);
        tmpnum[0]  = '\0';
        k          = 0;

        /************************/
        /* PARSE OUT EACH INDEX */
        /************************/

        for (j=0; j <= indxlen; j++)
        {
            if ((j == indxlen) || (Nperr.nerr[i].errindx[j] == ' '))
            {
                tmpnum[k] = '\0';
                indxnum   = atoi(tmpnum);
```

```
093
094                 /*****************************/
095                 /* Write the error messages */
096                 /*****************************/
097
098                 (void) fprintf(Rfp, "%s", Errdict.msg[indxnum].em1);
099                 (void) fprintf(Ofp, "%s", Errdict.msg[indxnum].em1);
100
101                 if ((strlen(Errdict.msg[indxnum].em2)) > 0)
102                 {
103                     (void) fprintf(Rfp, "%s", Errdict.msg[indxnum].em2);
104                     (void) fprintf(Ofp, "%s", Errdict.msg[indxnum].em2);
105                 }
106                 k = 0;
107             }
108             else
109             {
110                tmpnum[k] = Nperr.nerr[i].errindx[j];
111                k++;
112             }
113         }
114
115         (void) fprintf(Rfp, "\n");
116         (void) fprintf(Ofp, "\n");
117     }
118     (void) fclose(Efp);
119     (void) fclose(Ifp);
120
121
122     /*****************************************************************/
123     /* ATTEMPT TO COMPARE ERRORS IN THIS PACKAGE AGAINST LAST PACKAGE */
124     /*****************************************************************/
125
126     if ((su_package == FALSE) || (diff == FALSE))
127     {
128         goto end_run;
129     }
130
131     if ((src_dfp = opendir(LASTEDIR)) == NULL)
```

```
132       {
133           /* Can't compare last package directory doesn't exist */
134
135           (void) fprintf(Ofp,
136                         "\nERROR: Directory ./%s does not exist!", LASTEDIR);
137
138           (void) fprintf(Ofp,
139                         "No comparsion between this package run and the last package can be
140   made\n");
141
142           (void) fprintf(Rfp,  "
```

```
\nERROR: Directory ./%s does not exist!\n",
                                LASTEDIR);
         (void) fprintf(Rfp,  "       Attempting to create it for next run\n");

         /* Copy the current error file to the last_perr directory */

         (void) strcpy(tbuf, LASTEDIR);
            if ((i = mknod(tbuf, 0040000, NULL)) == -1)
         {
            (void) fprintf(Rfp,  "Attempt mkdir %s FAILED with errno=%d\n",
                                  tbuf, errno);
         }
         else
         {
             /* Set the permissions */

             if ((i = chmod(tbuf, 0755)) == -1)
             {
                 (void) fprintf(Rfp,
                                "Error - failed to change directory permissions - errno=%d\n",
                                errno);
             }
             (void) sprintf(tbuf, "cp %s %s/%s",
                                   Err_file, LASTEDIR, Err_file);
             (void) system(tbuf);
         }


         (void) fprintf(Rfp,
                        "No comparsion between this package run and the last package can be
made\n");
         goto end_run;
      }

      /***********************************/
      /* LOOK FOR LAST PACKAGE ERROR FILE */
      /***********************************/

      /* Copy the first two charcters of the pack_name so that our search   */
```

```
182     /* will pick up a previous package of the same generic to compare     */
183     /* against.                                                           */
184
185     (void) strncpy(file_Lpfx, pack_name, 2);
186     (void) strcpy (file_sfx, ERRSUFFIX);
187
188     /*  The second character of the pack_name is the loadid identifier    */
189     /*  We now allow 3000 package SU's to be applied as upper case.       */
190     /*  As an example:    BWM97-3Z07.  The package name would be 3Z07.    */
191     /*  The loadid would be: 'Z'.  Because we can't guarantee that the    */
192     /*  last package had the same case of this package, we need to check  */
193     /*  for both upper and lower case names so that each package can be   */
194     /*  upper case or lower case and tools still work.                    */
195
196     (void) strncpy(file_Upfx, pack_name, 2);
197     file_Upfx[2] = toupper(file_Lpfx[2]);
198
199     last_errf[0] = '\0';
200     while ((src_dp = readdir(src_dfp)) != NULL)
201     {
202         if ( (src_dp->d_name[0] == '.') || (src_dp->d_ino == 0) )
203         {
204             /* Skip current, parent or deleted entries */
205
206             continue;
207         }
208
209         if  ( (((i = sindex(src_dp->d_name, file_Lpfx)) != NOSTRFOUND) ||
210                ((i = sindex(src_dp->d_name, file_Upfx)) != NOSTRFOUND))
211             && ((j = sindex(src_dp->d_name, file_sfx))  != NOSTRFOUND)  )
212         {
213             /* We found a previous package error file   */
214             /* Do we have multiple files?  If so, abort */
215
216             if (last_errf[0] != '\0')
217             {
218                 (void) fprintf(stderr,
219                                "\nMultiple previous error files exist!\n");
220
```

```
                (void) fprintf(stderr, "Files [%s] and [%s] were found\n",
                                        last_errf, src_dp->d_name);
                (void) fprintf(stderr,
                                "\nNo comparsion between this package run and the last package can
be made\n");

                goto end_run;
            }
            (void) strcpy(last_errf, src_dp->d_name);

            /* Copy the last package name */

            (void) strncpy(lpack_name, src_dp->d_name, j-1);
            break;
        }

    }   /* End of while */

    (void) closedir(src_dfp);

    if (last_errf[0] == '\0')
    {
        (void) fprintf(stderr,
                        "\nNo Previous error files exist in [%s]", LASTEDIR);
        (void) fprintf(stderr,
                        "\nNo comparsion between this package run and the last package can be
made\n");

        /* Copy the current error file to the last_perr directory */

        (void) sprintf(tbuf, "cp %s %s/%s", Err_file, LASTEDIR, Err_file);
        (void) system(tbuf);

        goto end_run;
    }

    /* Align the two comparison lines -- For SU  Package: xxxx */
    /*                                   vs. Package: yyyy */

```

```
260         pos = sindex(Inbuf[2], HDRSTR1);
261         for (i=0; i<pos-1; i++)
262             linpkg[i] = ' ';
263         linpkg[i] = '\0';
264 
265         (void) strcat (linpkg, HDRSTR2);
266         (void) strcat (linpkg, " ");
267         (void) strcat (linpkg, lpack_name);
268 
269         /*******************************/
270         /* READ THE PREVIOUS ERROR FILE */
271         /*******************************/
272 
273         (void) sprintf(tbuf, "%s/%s", LASTEDIR, last_errf);
274         if ((Ifp = fopen(tbuf, "r")) == NULL)
275         {
276             (void) fprintf(stderr,
277                            "\nError: Can't Open Previous Error File: %s\n",
278                            last_errf);
279             exit(1);
280         }
281 
282         /* Continuously read the previous error file and build error indexes  */
283         /* Read error messages into the Inbuf array starting at position 10   */
284         /* Ten was chosen arbitrarily because we need to re-use the Inbuf     */
285         /* front portion of the array as this contains heading info for       */
286         /* printing our headings.                                             */
287 
288         i = lineno = 10;
289 
290         /* set the package flag = 0 (previous package errors) */
291         /*     and the offset to the beginning of the errors  */
292 
293         pkgflag = 0;
294         off     = 0;
295 
296         while ((strptr1 = fgets(Inbuf[lineno], ERRMSGSZ-1, Ifp)) != NULL)
297         {
298             if ((strcmp(Inbuf[lineno], BLANKS)) == 0)
```

```
299         {
300             /* Blank line - your at the end of the error message cluster */
301
302             bldindx(i, lineno-1, off, pkgflag);
303             lineno = 10;
304         }
305            else
306         {
307             lineno++;
308         }
309     }
310     (void) fclose(Ifp);
311
312     /*************************************************************/
313     /* COMPARE THE PREVIOUS UNIQUE ERRORS WITH THE CURRENT ERRORS */
314     /*************************************************************/
315
316
317     /* For each error in the current file, loop through the previous file */
318     /* to see if the error also occurred in the previous file.            */
319
320     lineno = 0;
321        pageno = 0;
322     newcnt = 0;
323
324     /* Build header line Inbuf[0] */
325
326     if (caption[0] != '\0')
327     {
328         (void) sprintf(thdr0,
329                        "NEW Application %s Error Messages", caption);
330     }
331     else
332     {
333         (void) sprintf(thdr0,
334                        "NEW Application Error Messages");
335     }
336
337     /* Build header line Inbuf[3] with a line of equal signs */
```

```
338
339         i = strlen(thdr0);
340         for (j=0; j<i; j++)
341             thdr3[j] = '=';
342         thdr3[j] = '\0';
343
344         /* center header line 1 */
345
346         capcntr(RPTWIDTH, thdr0);
347
348         /* center header line 3 */
349
350         capcntr(RPTWIDTH, thdr3);
351         (void) sprintf(Inbuf[3], "%s\n\n", thdr3);
352
353         /* Prepare header line 1 to contain the Page % literal 8 columns      */
354         /* from the end of the page width by filling with blanks to that point*/
355
356         i = strlen(thdr0);
357
358         for (j=i; j < RPTWIDTH-8; j++)
359             thdr0[j] = ' ';
360         thdr0[j] = '\0';
361
362         /* Modify Inbuf[2] to contain 2 lines */
363
364         (void) strcat(Inbuf[2], linpkg);
365         (void) strcat(Inbuf[2], "\n");
366
367
368         /*************************************************/
369         /* LOOP THROUGH BOTH CURRENT (i) AND OLD (j) ERRORS */
370         /*************************************************/
371
372         for (i=1;  i <= Nperr.entries; i++)
373         {
374             pos = 0;
375             for (j=1;  j <= Operr.entries; j++)
376             {
```

```
377
378             if ((strcmp(Nperr.nerr[i].errindx, Operr.oerr[j].errindx)) == 0)
379             {
380                 /* Error message is in both files - set pos to indicate so*/
381
382                 pos = j;
383                 break;
384             }
385         }
386         if (pos == 0)
387         {
388             /*********************/
389             /* NEW ERROR MESSAGE */
390             /*********************/
391
392             lineno += Nperr.nerr[i].no_lines + 2;
393
394             if ((pageno == 0) || (lineno >= 59))
395             {
396                 pageno++;
397                 (void) sprintf(Inbuf[0], "
```

```
\n%s Page %d\n",
                                thdr0, pageno);

                /* Use Inbuf[1] from previous page */
                /*     Inbuf[2] modified above     */

                for (k=0; k<4; k++)
                {
                    (void) fprintf(Rfp, "%s", Inbuf[k]);
                    (void) fprintf(Ofp, "%s", Inbuf[k]);
                }
                lineno = 7 ;
                lineno += Nperr.nerr[i].no_lines + 2;
            }

            /* Loop through the errindx to pull out each error message    */
            /* out of the error dictionary.  A space will seperate each   */
            /* index.                                                     */

            indxlen    = strlen(Nperr.nerr[i].errindx);
            tmpnum[0]  = '\0';
            m          = 0;

            /* Add one to our new error count */
            newcnt++;

            (void) fprintf(Ofp, "%5d. NEW MESSAGE OCCURRED: %5d
.........................................\n",
                                newcnt, Nperr.nerr[i].no_entries);
            (void) fprintf(Rfp, "%5d. NEW MESSAGE OCCURRED: %5d
.........................................\n",
                                newcnt, Nperr.nerr[i].no_entries);
            /************************/
            /* PARSE OUT EACH INDEX */
            /************************/

            for (k=0; k <= indxlen; k++)
            {
                if ((k == indxlen) || (Nperr.nerr[i].errindx[k] == ' '))
```

```
437                 {
438                     tmpnum[m] = '\0';
439                     indxnum   = atoi(tmpnum);
440 
441                     /*******************************/
442                     /* WRITE THE NEW ERROR MESSAGES */
443                     /*******************************/
444 
445                     (void) fprintf(Ofp, "%s", Errdict.msg[indxnum].em1);
446                     (void) fprintf(Rfp, "%s", Errdict.msg[indxnum].em1);
447 
448 
449                     if ((strlen(Errdict.msg[indxnum].em2)) > 0)
450                     {
451                         (void) fprintf(Ofp, "%s", Errdict.msg[indxnum].em2);
452                         (void) fprintf(Rfp, "%s", Errdict.msg[indxnum].em2);
453                     }
454                     m = 0;
455                 }
456                 else
457                 {
458                    tmpnum[m] = Nperr.nerr[i].errindx[k];
459                    m++;
460                 }
461             }
462 
463             (void) fprintf(Ofp, "\n");
464             (void) fprintf(Rfp, "\n");
465 
466         }   /* end if pos == 0 */
467 
468     }
469 
470     /*************************************************/
471     /* Print out the final statistics for the comparison */
472     /*************************************************/
473 
474     lineno = lineno + 8;
475     if ((pageno == 0) || (lineno >= 59))
```

```
476         {
477             pageno++;
478             (void) sprintf(Inbuf[0], "
```

```
\n%s Page %d\n",
                        thdr0, pageno);

        /* Use Inbuf[1] from previous page */
        /*     Inbuf[2] modified above     */

        for (k=0; k<4; k++)
        {
            (void) fprintf(Rfp, "%s", Inbuf[k]);
            (void) fprintf(Ofp, "%s", Inbuf[k]);
        }
        lineno = 15 ;
    }

    (void) fprintf(Rfp,
           "                        *** END OF PACKAGE COMPARISON ***\n");

    (void) fprintf(Rfp,
           "\n\nCURRENT  Package: %s has   %4d UNIQUE error messages\n\n",
            pack_name, Nperr.entries);

    (void) fprintf(Rfp,
           "CURRENT  Package: %s ADDED %4d NEW    error messages\n\n",
            pack_name, newcnt);

    (void) fprintf(Rfp,
           "PREVIOUS Package: %s had   %4d UNIQUE error messages\n",
            lpack_name, Operr.entries);

    (void) fprintf(Ofp,
           "                        *** END OF PACKAGE COMPARISON ***\n");

    (void) fprintf(Ofp,
           "\n\nCURRENT  Package: %s has   %4d UNIQUE error messages\n\n",
            pack_name, Nperr.entries);

    (void) fprintf(Ofp,
           "CURRENT  Package: %s ADDED %4d NEW    error messages\n\n",
            pack_name, newcnt);
```

```
518
519       (void) fprintf(Ofp,
520              "PREVIOUS Package: %s had   %4d UNIQUE error messages\n",
521               lpack_name, Operr.entries);
522
523       /*****************************************************/
524       /* Copy the current error file to last_perr directory */
525       /*****************************************************/
526
527       (void) sprintf(tbuf, "cp %s %s/%s", Err_file, LASTEDIR, Err_file);
528       (void) system(tbuf);
529
530       /*****************************************************/
531       /* Move the old error file to the old_perr directory */
532       /*****************************************************/
533
534       if ((src_dfp = opendir(OLDEDIR)) == NULL)
535       {
536           /* Directory doesn't exist - create it */
537
538           (void) strcpy(tbuf, OLDEDIR);
539
540           /* Disregard error message if directory already notice exists */
541
542              (void) mknod(tbuf, 0040000, NULL);
543       }
544       else
545       {
546           (void) closedir(src_dfp);
547       }
548
549       (void) sprintf(tbuf, "mv %s/%s %s/%s",
550                            LASTEDIR, last_errf, OLDEDIR, last_errf);
551       (void) system(tbuf);
552
553
554            /**************/
555  end_run:  /* END OF RUN */
556            /**************/
```

```
557
558
559       (void) fprintf(stdout,
560                      "\n\nAnalysis Complete - Check the following files -\n\n");
561       (void) fprintf(stdout,
562                      "  Output File (full run)   : %s\n", Out_file);
563       (void) fprintf(stdout,
564                      "  Report File (stats only) : %s\n", Rpt_file);
565       (void) fprintf(stdout,
566                      "  Unique Error Message File: %s\n", Err_file);
567       return(0);
568   }
569
570   /* EOF - TSTmain */
571
572
573
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:                sindex()
 *
 * Abstract:            Find the location within search_str where input_str
 *                      exists.
 *
 *    Programmer(s):    G. R. Daly
 *          Company:    TBD
 *             Date:    Mon Apr 26 07:45:00 CST
 *
 *
 * Loadable package:    BASE
 *
 * Usage:               (int) RCmindex(search_str, input_str)
 *
 * Parameters:          search_str - character string to search.
 *                      input_str  - input string to search for in search_str.
 *
 * Externals:           None.
 *
 * Returns:             Return Value(s) -
 *
 *                      Postion within search_str where input_str was found
 *                      OR
 *                      Zero to indicate input_str was not found in search_str.
 *
 * Description:         This function will search for the occurance of a given
 *                      input character string within another searched string.
 *                      If the input string exists within the searched string,
 *                      then the postion where the input string was found in the
 *                      searched string is returned.  Otherwise zero is returned
 *                      to indicate that the input string was not found.
 *
 *
```

```
613    * Warnings:           None.
614    *
615    * Examples:           None.
616    *
617    * Calls:              strlen()      - system string function
618    *
619    * Macros:             None.
620    *
621    *
622    * See Also:           Supporting documentation section.
623    *
624    ************************************************************************/
625
626
```

```
627
628  /* BOF - sindex */
629
630  int
631  sindex(search_str, input_str)
632  char *search_str;        /* string which we will search for input_str  */
633  char *input_str ;        /* input string used to search search_str     */
634  {
635       /* Local Variables */
636
637       short slen;         /* variable to hold length of searched string */
638       short mlen;         /* variable to hold length of input    string */
639       short i;       /* temporary index variable                   */
640       short j;       /* temporary index variable                   */
641       short match;        /* variable to record number of matches       */
642
643       /* BEGIN EXECUTABLE CODE */
644
645          /* Determine lengths of search and input strings */
646
647          slen = strlen(search_str);
648          mlen = strlen(input_str);
649
650          /* Loop through search_str until the number of matches */
651          /* equals the length of input_str OR until your can't  */
652          /* traverse through the search_str any longer.         */
653
654          for (i=0; i <= (slen - mlen); i++)
655          {
656                  match = 0;
657                  for (j=0; j < mlen; j++)
658                  {
659                          if (search_str[i+j] == input_str[j])
660                          {
661                                  match++;
662                                  if (match == mlen)
663                                          return(i+1);
664                          }
665                          else
```

```
                        {
                                match = 0;
                                j = mlen;
                        }
                }
        }
        return(NOSTRFOUND);

}

/* EOF - sindex */
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:               prcs_err()
 *
 * Abstract:           This function will store errors in the error dictionary
 *                     and write errors to various output files.
 *
 *    Programmer(s):   G. R. Daly
 *
 *          Company:   TBD
 *             Date:   Mon Apr 26 07:45:00 CST
 *
 *
 * Loadable package:   None.
 *
 * Usage:              int prcs_err(lineno, emode, eclass, eview,
 *                                  sname, tname, inpkg, elastscreen,
 *                                  errstart, errstop)
 *
 * Parameters:         lineno      - position within Inbuf[]
 *                     emode       - error screen mode
 *                     eclass      - error form class
 *                     eview       - error form view
 *                     sname       - error script name
 *                     tname       - error script test id number
 *                     iinpkg      - input package name to program
 *                     elastscreen - error last screen line
 *                     errstart    - postion within Inbuf[] where errors start
 *                     errstop     - postion within Inbuf[] where errors stop
 *
 *
 * Externals:          None.
 *
 *
 * Returns:            Return Value(s) -
```

```
718    *
719    *                    FAIL (1)   - exit with return code 1 and displays
720    *                                 an error message.
721    *
722    *
723    * Description:       This function examines an application output file.
724
725    *
726    * Implementation:    Standard UNIX.
727    *
728    *
729    * Warnings:          None.
730    *
731    * Examples:          None.
732    *
733    * Calls:             sprintf()   - Formats characters strings
734    *                    strcpy()    - copies one string to another
735    *                    strcmp()    - compares one character string to another
736    *                    fclose()    - closes stream files
737    *                    fprintf()   - Formats stream output
738    *                    exit()      - exits program
739    *                    sindex()    - looks for character strings
740    *
741    * Macros:            None.
742    *
743    *
744    * See Also:          Application and Verify Usage Guide.
745    *
746    ***************************************************************************/
747
```

```
748
749  /* BOF - prcs_err */
750
751  void
752  prcs_err(lineno, emode, eclass, eview, sname, tname, inpkg, elastscreen,
753           errstart, errstop)
754
755  int    lineno;                    /* position within Inbuf[]    */
756  char  *emode;                     /* error screen mode          */
757  short  eclass;                    /* error form class           */
758  short  eview;                     /* error form view            */
759  char  *sname;                     /* error script name          */
760  char  *tname;                     /* error script test id number*/
761  char  *inpkg;                     /* input package name         */
762  char  *elastscreen;               /* error last screen line     */
763  int    errstart;                  /* error start pos in Inbuf   */
764  int    errstop;                       /* error stop  pos in Inbuf   */
765  {
766       /* System Calls */
767
768       FILE *fopen();               /* open a stream file         */
769
770       extern int   strcmp();            /* compares character strings */
771
772       extern char *strcpy();            /* copies character strings   */
773
774       /* Local Calls */
775
776       int   sindex();                   /* looks for character strings*/
777       void  bldindx();             /* builds error msg indexes   */
778
779       /* Local Variables */
780
781       int     i;                   /* index variable             */
782       short   pkgflag;             /* prev or curr package flag  */
783       short   off;                 /* offset from the beginning  */
784                                 /* Inbuf where the error      */
785                                 /* starts                     */
786
```

```
787         char    cur_vname[20];             /* current   error view name  */
788         static  char last_vname[20] = "NONE";   /* last open error view name  */
789
790         char    cur_sname[20];             /* current   error script name*/
791         static  char last_sname[20] = "NONE";   /* last open error script name*/
792
793
794         /* BEGIN EXECUTABLE CODE */
795
796         /* For the error messages just received, build the error indexes and  */
797         /* insert the unique error messages into the error dictionary         */
798
799
800         /* Calculate the error message starting offset */
801
802         if ((strcmp(emode, SPASS)) == 0)
803             off = 17;
804         else
805         {
806             /* Determine the offset for expected FAIL's (and passed) to be    */
807             /* the first alphabetic character found.                          */
808
809             for (i=0; i<strlen(Inbuf[errstart]); i++)
810             {
811                 if ( ((Inbuf[errstart][i] >= 'a')&&(Inbuf[errstart][i] <= 'z'))
812                   || ((Inbuf[errstart][i] >= 'A')&&(Inbuf[errstart][i] <= 'Z')))
813                 {
814                      break;
815                 }
816             }
817
818             if (i == strlen(Inbuf[errstart]))
819             {
820                     (void) fprintf(stderr,
821                                    "Error: Can't find the start of the error message: \n%s\n",
822                                     Inbuf[errstart]);
823                     (void) fclose(Ofp);
824                     (void) fclose(Ifp);
825
```

```
826                 exit(1);
827             }
828             else
829                 off = i;
830         }
831
832         /* set the package flag = 1 (current package errors) */
833         pkgflag = 1;
834
835         bldindx(errstart, errstop, off, pkgflag);
836
837
838         /*********************************************************/
839         /* WRITE THE ERROR MESSAGES TO THE USER SELECTED LOG FILES */
840         /*********************************************************/
841
842
843         if (Vlog == TRUE)
844         {
845             /*****************************/
846             /* LOG ERRORS BY VIEW NUMBER */
847             /*****************************/
848
849
850             /* Create the string to contain the current error file name */
851
852             if (eclass == 0)
853                 (void) sprintf(cur_vname, "E.a.%d", eview);
854
855             else if (eclass == 1)
856                 (void) sprintf(cur_vname, "E.b.%d", eview);
857
858             else (void) sprintf(cur_vname, "E.%d.%d",  eclass-2, eview);
859
860             /* Check if you have to open a new file */
861
862             if ((strcmp(cur_vname, last_vname)) != 0)
863             {
864                 /* Close the previous file desciptor */
```

```
865             /* if it isn't the first time        */
866
867             if ((strcmp(last_vname, "NONE")) != 0)
868             {
869                 (void) fclose(Vfp);
870             }
871
872             if ((Vfp = fopen(cur_vname, "a+")) == NULL)
873             {
874                 (void) fprintf(stderr,
875                                "Error: fopen() Can't Append to VIEW ERROR File: %s\n",
876                                 cur_vname);
877                 (void) fclose(Ofp);
878                 (void) fclose(Ifp);
879                 exit(1);
880             }
881
882             /* Copy the current file name open for the next time */
883
884             (void) strcpy(last_vname, cur_vname);
885         }
886
887
888         /***************************/
889         /* Display the RC in error */
890         /***************************/
891
892             /* DISPLAY ERROR HEADINGS */
893
894         (void) fprintf(Vfp, SCREEN1FMT, emode);
895
896         if (tname[0] == '\0')
897         {
898            (void) fprintf(Vfp,
899                           "%-25s                                        %s\n\n",
900                               cur_vname, inpkg);
901         }
902         else
903         {
```

```
904             (void) fprintf(Vfp,
905                            "%-25s  Test Id: %-12s             %s\n\n",
906                                cur_vname, tname, inpkg);
907         }
908
909         /* Display the last screen request */
910
911         (void) fprintf(Vfp, "%s", elastscreen);
912
913
914         /* Loop through the error Application */
915
916         for (i=0; i<=lineno; i++)
917         {
918             (void) fprintf(Vfp, "%s", Inbuf[i]);
919         }
920
921         (void) fprintf(Vfp, "\n%s\n", DASHES);
922
923     }   /* end if Vlog == TRUE (log errors by view number) */
924
925
926     else if (Slog == TRUE)
927     {
928         /*****************************/
929         /* LOG ERRORS BY SCRIPT NAME */
930         /*****************************/
931
932
933         (void) sprintf(cur_sname, "E.%s", sname);
934
935         /* Check if you have to open a new file */
936
937         if ((strcmp(cur_sname, last_sname)) != 0)
938         {
939             /* Close the previous file desciptor */
940             /* if it isn't the first time        */
941
942             if ((strcmp(last_sname, "NONE")) != 0)
```

```
943             {
944                 (void) fclose(Sfp);
945             }
946
947             if ((Sfp = fopen(cur_sname, "a+")) == NULL)
948             {
949                 (void) fprintf(stderr,
950                                "Error: fopen() Can't Append to SCRIPT ERROR File: %s\n",
951                                 cur_sname);
952                 (void) fclose(Ofp);
953                 (void) fclose(Ifp);
954                 exit(1);
955             }
956
957             /* Copy the current file name open for the next time */
958
959             (void) strcpy(last_sname, cur_sname);
960         }
961
962         /**************************/
963         /* Display the RC in error */
964         /**************************/
965
966             /* DISPLAY ERROR HEADINGS */
967
968         (void) fprintf(Sfp, SCREEN1FMT, emode);
969
970         if (tname[0] == '\0')
971         {
972            (void) fprintf(Sfp,
973                           "%-25s                                        %s\n\n",
974                               sname, inpkg);
975         }
976         else
977         {
978            (void) fprintf(Sfp,
979                           "%-25s  Test Id: %-12s             %s\n\n",
980                               sname, tname, inpkg);
981         }
```

```
982
983         /* Display the last screen request */
984
985         (void) fprintf(Sfp, "%s", elastscreen);
986
987
988         /* Loop through the error Application */
989
990         for (i=0; i<=lineno; i++)
991         {
992             (void) fprintf(Sfp, "%s", Inbuf[i]);
993         }
994
995         (void) fprintf(Sfp, "\n%s\n", DASHES);
996
997     }   /* end if Slog == TRUE (log errors by script name) */
998
999
1000    /* Output file error logging is allowed even if the user selected     */
1001    /* error logging by either view number or script name.                */
1002
1003    if (Olog == TRUE)
1004    {
1005        /*****************************/
1006        /* LOG ALL ERRORS IN ONE FILE */
1007        /*****************************/
1008
1009
1010            /* DISPLAY ERROR HEADINGS */
1011
1012        (void) fprintf(Ofp, SCREEN1FMT, emode);
1013
1014        if (tname[0] == '\0')
1015        {
1016           (void) fprintf(Ofp,
1017                          "%-25s                                    %s\n\n",
1018                              sname, inpkg);
1019        }
1020        else
```

```
021           {
022              (void) fprintf(Ofp,
023                             "%-25s  Test Id: %-12s             %s\n\n",
024                                 sname, tname, inpkg);
025           }
026
027           /* Display the last screen request */
028
029           (void) fprintf(Ofp, "%s", elastscreen);
030
031
032           /* Loop through the error Application */
033
034           for (i=0; i<=lineno; i++)
035           {
036               (void) fprintf(Ofp, "%s", Inbuf[i]);
037           }
038
039           (void) fprintf(Ofp, "\n%s\n", DASHES);
040       }
041
042
043       return;
044   }
045
046   /* EOF - prcs_err */
047
048
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:               crheap()
 *
 * Abstract:           This function will create a heap from an array of
 *                     characters.
 *
 *    Programmer(s):   G. R. Daly
 *
 *          Company:   TBD
 *             Date:   Mon Apr 26 07:45:00 CST
 *
 *
 * Loadable package:   None.
 *
 * Usage:              void crheap(no_entries)
 *
 *
 * Parameters:         no_entries  - number of entries in heap.
 *
 * Externals:          None.
 *
 *
 * Returns:            Return Value(s) -
 *
 *                     NONE.
 *
 *
 * Description:        This function will modify an existing array of
 *                     in order to perform a heap sort on them.
 *
 * Implementation:     Standard UNIX.
 *
 *
 * Warnings:           None.
```

```
088     *
089     * Examples:           None.
090     *
091     * Calls:              strcmp()    - compares one character string to another
092     *                     strcpy()    - copies one string to another
093     *
094     * Macros:             None.
095     *
096     *
097     * See Also:           Not Applicable.
098     *
099     ***************************************************************************/
100
101
```

```
102
103  /* BOF - crheap */
104
105  void
106  crheap(no_entries)
107  int   no_entries;                 /* number of entries in heap  */
108  {
109       /* System Calls */
110
111       extern int   strcmp();             /* compares character strings */
112       extern char *strcpy();             /* copies character strings   */
113
114       /* Local Variables */
115
116       int   i;                  /* index variable             */
117       int   parent;                  /* index variable             */
118       int   insert;                  /* index variable             */
119
120       char  key_name[NAMESZ];             /* script name                */
121       int   key_count;               /* script name error count    */
122
123       /* BEGIN EXECUTABLE CODE */
124
125       /* Build a heap */
126
127       for (insert=2; insert<=no_entries; insert++)
128       {
129           i         = insert;
130           key_count = Rc[insert].count;
131
132           (void) strcpy(key_name, Rc[insert].name);
133
134           /* Obtain the parent of the next member of the array */
135
136           parent = i/2;
137
138           /* Place the new member in the existing heap */
139
140           while ( (i > 1) && ((strcmp(key_name, Rc[parent].name)) > 0) )
```

```
141             {
142                 /* Interchange members */
143
144                 (void) strcpy(Rc[i].name, Rc[parent].name);
145                 Rc[i].count = Rc[parent].count;
146
147                 /* Obtain next parent */
148
149                 i      = parent;
150                 parent = i/2;
151
152                 /* Check array boundary */
153
154                 if (parent < 1)
155                     parent = 1;
156
157             }   /* end of while */
158
159             /* Copy member into heap */
160
161             (void) strcpy(Rc[i].name, key_name);
162             Rc[i].count = key_count;
163
164
165         }   /* end of for */
166
167         return;
168     }
169
170     /* EOF - crheap */
171
172
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:               updstat()
 *
 * Abstract:           This function will update the statistics for tests
 *                     that pass or fail.
 *
 *    Programmer(s):   G. R. Daly
 *
 *          Company:   TBD
 *             Date:   Mon Apr 26 07:45:00 CST
 *
 *
 * Loadable package:   None.
 *
 * Usage:              void updstat(errflag, classno, viewno, lineno,
 *                                  operation, script_name)
 *
 *
 * Parameters:         errflag     - flag indicates if test failed
 *                     classno     - current form class
 *                     viewno      - current form view
 *                     lineno      - position within Inbuf array
 *                     operation   - current form operation
 *                     script_name - current script name (for failed type)
 *
 * Externals:          None.
 *
 *
 * Returns:            Return Value(s) -
 *
 *                     NONE.
 *
 *
 * Description:        This function will update the pass and failed
```

```
212    *                     statistics for passed and failed operations.
213    *
214    * Implementation:     Standard UNIX.
215    *
216    *
217    * Warnings:           None.
218    *
219    * Examples:           None.
220    *
221    * Calls:              strcmp()    - compares one character string to another
222    *                     sindex()    - looks for character strings
223    *
224    * Macros:             None.
225    *
226    *
227    * See Also:           Not Applicable.
228    *
229    ***************************************************************************/
230
231
```

```
232
233  /* BOF - updstat */
234
235  void
236  updstat(errflag, classno, viewno, lineno, operation, script_name)
237  short errflag;                    /* did the current rc fail    */
238  short classno;                    /* current form class         */
239  short viewno;                     /* current form views         */
240  int   lineno;                     /* position within Inbuf      */
241  char *operation;                  /* current form operatione    */
242  char *script_name;                /* error script name          */
243  {
244       /* System Calls */
245
246       extern int   strcmp();            /* compares character strings */
247       extern char *strcpy();            /* copies character strings   */
248
249       /* Local Calls */
250
251       int   sindex();                   /* looks for character strings*/
252
253       /* Local Variables */
254
255       int   pos;                   /* index variable             */
256       int   i;                /* index variable             */
257       int   k;                /* index variable             */
258
259       /* BEGIN EXECUTABLE CODE */
260
261       /* Determine array index offset - Have A and B views stored first  */
262       /* so that they appear to be arranged lexigraphically when printed */
263
264       if (errflag == FALSE)
265       {
266           /***************/
267           /* TEST PASSED */
268           /***************/
269
270           /*********************************/
```

```
271         /* Update Form Operation Attempt */
272         /*********************************/
273
274         if ((strcmp(operation, NEW)) == 0)
275         {
276             Class[classno].View[viewno].Pass.new++;
277             Pnew++;
278         }
279
280         else if ((strcmp(operation, CHG)) == 0)
281         {
282             Class[classno].View[viewno].Pass.chg++;
283             Pchg++;
284         }
285
286         else if ((strcmp(operation, OUT)) == 0)
287         {
288             Class[classno].View[viewno].Pass.out++;
289             Pout++;
290         }
291
292         else if ((strcmp(operation, VFY)) == 0)
293         {
294             Class[classno].View[viewno].Pass.vfy++;
295             Pvfy++;
296         }
297
298         else if ((strcmp(operation, MVFY)) == 0)
299         {
300             Class[classno].View[viewno].Pass.mvfy++;
301             Pmvfy++;
302         }
303
304         else if ((strcmp(operation, SHVFY)) == 0)
305         {
306             Class[classno].View[viewno].Pass.shvfy++;
307             Pshvfy++;
308         }
309
```

```
310         else if ((strcmp(operation, ATTR)) == 0)
311         {
312             Class[classno].View[viewno].Pass.attr++;
313             Pattr++;
314         }
315
316         else
317         {
318             (void) fprintf(stderr,
319                         "\nupdstat(): PASS: unknown %d.%d operation [%s]\n",
320                             classno, viewno, operation);
321             Punkwn++;
322             Class[classno].View[viewno].Pass.unkwn++;
323
324             for (k=0; k<=lineno; k++)
325             {
326                 (void) fprintf(stderr,
327                             "Inbuf[%d]: %s", k, Inbuf[k]);
328             }
329             (void) fprintf(stderr, "\n");
330
331         }
332
333        Class[classno].View[viewno].Pass.count++;
334        Tstpass++;
335      }
336      else
337      {
338         /***************/
339         /* TEST FAILED */
340         /***************/
341
342         /*********************************/
343         /* Update Form Operation Attempt */
344         /*********************************/
345
346         if ((strcmp(operation, NEW)) == 0)
347         {
348             Class[classno].View[viewno].Fail.new++;
```

```
            Fnew++;
        }

        else if ((strcmp(operation, CHG)) == 0)
        {
            Class[classno].View[viewno].Fail.chg++;
            Fchg++;
        }

        else if ((strcmp(operation, OUT)) == 0)
        {
            Class[classno].View[viewno].Fail.out++;
            Fout++;
        }

        else if ((strcmp(operation, VFY)) == 0)
        {
            Class[classno].View[viewno].Fail.vfy++;
            Fvfy++;
        }

        else if ((strcmp(operation, MVFY)) == 0)
        {
            Class[classno].View[viewno].Fail.mvfy++;
            Fmvfy++;
        }

        else if ((strcmp(operation, SHVFY)) == 0)
        {
            Class[classno].View[viewno].Fail.shvfy++;
            Fshvfy++;
        }

        else if ((strcmp(operation, ATTR)) == 0)
        {
            Class[classno].View[viewno].Fail.attr++;
            Fattr++;
        }

```

```
388         else
389         {
390             (void) fprintf(stderr,
391                           "\nupdstat(): FAIL: unknown %d.%d operation [%s]\n",
392                               classno, viewno, operation);
393             Funkwn++;
394             Class[classno].View[viewno].Fail.unkwn++;
395 
396             for (k=0; k<=lineno; k++)
397             {
398                 (void) fprintf(stderr,
399                               "Inbuf[%d]: %s", k, Inbuf[k]);
400             }
401 
402             (void) fprintf(stderr, "\n");
403 
404         }
405 
406         Class[classno].View[viewno].Fail.count++;
407         Tstfail++;
408 
409         /****************************/
410         /* Update RC/V Script Error */
411         /****************************/
412 
413         /* Don't use the first position in the array */
414 
415         for (k=1; k<=Serr_cnt; k++)
416         {
417             pos = sindex(Rc[k].name, script_name);
418             if (pos != NOSTRFOUND)
419             {
420                 Rc[k].count = Rc[k].count + 1;
421                 break;
422             }
423         }
424 
425         if ( (k > Serr_cnt) || (Serr_cnt == 0))
426         {
```

```
427                 Serr_cnt++;
428
429                 if (Serr_cnt > MAXSCRERR)
430                 {
431                     (void) fprintf(stderr,
432                                 "\nError:  Scripts in error exceed maximum array size\n");
433                     (void) fprintf(stderr,
434                                 "ARRAY CONTAINS THE FOLLOWING SCRIPTS:\n");
435
436                     for (i=0; i<MAXSCRERR; i++)
437                         (void) fprintf(stderr, "%4d. %s\n", i, Rc[i].name);
438
439                     (void) fclose(Ofp);
440                     (void) fclose(Ifp);
441                     exit(1);
442
443                 }
444                 (void) strcpy(Rc[Serr_cnt].name, script_name);
445                 Rc[Serr_cnt].count++;
446             }
447
448         }
449
450
451         return;
452     }
453
454     /* EOF - updstat */
455
456
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:               capcntr()
 *
 * Abstract:           This function will center a string of characters for
 *                     reports to a page.
 *
 *    Programmer(s):   G. R. Daly
 *          Company:   TBD
 *             Date:   Wed Apr 20 15:54:00 CST
 *
 *
 * Loadable package:   BASE
 *
 * Usage:              (void) capcntr(line_size, input_str)
 *
 * Parameters:         line_size  - length of the line you want to center on
 *                     input_str  - input string to be centered.
 *
 * Externals:          None.
 *
 * Returns:            Return Value(s) -
 *
 *                     None.
 *
 * Description:        This function will center the input string according
 *                     to the line size specified.  If the string length is
 *                     greater then the line size, nothing will be done.
 *
 *
 * Warnings:           None.
 *
 * Examples:           None.
 *
 * Calls:              strlen()    - system string function
```

```
496     *                       strcat()    - concatenates one string to another
497     *
498     * Macros:               None.
499     *
500     *
501     * See Also:             Supporting documentation section.
502     *
503     ******************************************************************************/
504
505
```

```
506
507  /* BOF - capcntr */
508
509  void
510  capcntr(line_size, input_str)
511  int   line_size;              /* size of the line input is to be centered   */
512  char *input_str ;        /* input string used to search search_str     */
513  {
514       /* System Calls */
515
516       extern char *strcpy();             /* copies character strings   */
517       extern char *strcat();             /* concatenates strings       */
518
519       /* Local Variables */
520
521       int   i;                 /* temporary index            */
522       int   pos;                    /* temporary index            */
523       int   buflen;                 /* buffer length of input     */
524
525       char  temp_str[MAXREAD];
526
527       /* BEGIN EXECUTABLE CODE */
528
529       /* Test for invalid line_size */
530
531       if ((line_size < 0) || (line_size >= MAXREAD))
532           return;
533
534       (void) strcpy(temp_str, input_str);
535
536       buflen = strlen(temp_str);
537
538       /* Test for input string greater then line_size */
539
540       if (buflen >= line_size)
541           return;
542
543       pos = line_size - strlen(temp_str);
544
```

```
545       if (pos <= 0)
546           return;
547
548       pos = pos/2;
549
550       /* Add blank spaces for centering */
551
552       for (i=0; i<pos; i++)
553           input_str[i] = ' ';
554
555       input_str[i] = '\0';
556
557       (void) strcat(input_str, temp_str);
558
559       return;
560
561   }
562
563   /* EOF - capcntr */
564
565
566
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:                bldindx()
 *
 * Abstract:            This function will build a string of error dictionary
 *                      indexes for each error message.
 *
 *    Programmer(s):    G. R. Daly
 *
 *          Company:    TBD
 *             Date:    Fri Oct 14 15:50:00 CST
 *
 *
 * Loadable package:    None.
 *
 * Usage:               bldindx(start, stop, emode, pkgflag)
 *
 *
 * Parameters:          start      - starting position of the error message
 *                                   within the Inbuf array.
 *                      stop       - stopping position of the error message
 *                                   within the Inbuf array.
 *                      off        - offset from the beginning of the Inbuf
 *                                   where the error message begins
 *                      pkgflag    - a flag when set to 1 = current  package
 *                                       or when set to 0 = previous package
 *
 * Externals:           None.
 *
 *
 * Returns:             Return Value(s) -
 *
 *                      FAIL (1)   - exit with return code 1 and displays
 *                                   an error message.
 *
```

```
606    *
607    * Description:        This function will repeatedly call inserr to get
608    *                     an error dictionary index and will store all the indexes
609    *                     in the applicable structure.
610    *
611    * Implementation:     Standard UNIX.
612    *
613    *
614    * Warnings:           None.
615    *
616    * Examples:           None.
617    *
618    * Calls:              strcpy()    - copies one string to another
619    *                     strcmp()    - compares one character string to another
620    *                     strcat()    - concatenates one string to another
621    *                     sprintf()   - Formats character strings
622    *                     exit()      - exits program
623    *                     fopen()     - opens a file for stream
624    *                     inserr()    - inserts error message into err dictionary
625    *
626    * Macros:             None.
627    *
628    *
629    * See Also:           Application Usage Guide.
630    *
631    ******************************************************************************/
632
633
```

```
/* BOF - bldindx */

void
bldindx(start, stop, off, pkgflag)
int   start;                  /* starting postion within Inbuf      */
int   stop;                   /* stopping postion within Inbuf      */
short off;                    /* offset from Inbuf[0] where error is*/
short pkgflag;                /* indicates current or previous pkg  */

{
     /* System Calls */

     FILE *fopen();               /* open a stream file        */

     extern int   strcmp();            /* compares character strings */

     extern char *strcpy();            /* copies character strings   */
     extern char *strcat();            /* concatenates strings       */

     /* Local Calls */

     int   inserr();                   /* inserts error message in   */

     /* Local Variables */

     int     i;                   /* index variable            */
     int     j;                   /* index variable            */
     int     pos;                 /* index variable            */

     int   indxlen;               /* length of index string    */
     int   indxnum;               /* index number              */
     int   no_eindx;                   /* number of error indexes    */
     int   no_elines;             /* number of printable errors */

     char  tmpindx[ERRMSGSZ+1];        /* temporary error index     */

     char  message1[ERRMSGSZ+1];          /* temporary error message    */
     char  message2[ERRMSGSZ+1];          /* temporary error message    */
     char  tmpnum[15];            /* temporary number buffer    */
```

```
673
674
675         /* BEGIN EXECUTABLE CODE */
676
677         /* Pick up each error message and store the message in the error      */
678         /* dictionary and build string containing the indexes into the error  */
679         /* dictionary for each error message in the cluster.                  */
680
681         no_eindx   = 0;
682         tmpindx[0] = '\0';
683
684         while (start <= stop)
685         {
686             /* Determine if the current error is 1 or 2     */
687             /* lines.  Two line error messages begin with   */
688             /* a number such as: "138-2 DATA BASE SYSTEM    */
689             /* ERROR" where one line error messages are:    */
690             /* "?E Form not found"                          */
691
692                        /* Application: a PASS expected */
693
694             /*       Inbuf postion  1-10 = the line number  */
695             /*       Inbuf postion    11 = a blank          */
696             /*       Inbuf postion 12-15 = MSG!             */
697             /*       Inbuf postion    16 = a blank          */
698             /*       Inbuf postion 17    = error message    */
699
700                        /* Application: a FAIL expected */
701
702             /*       Inbuf postion  1-10 = the line number  */
703             /*       Inbuf postion    11 = a blank          */
704             /*       Inbuf postion 12    = apptext input    */
705             /*                             for expect FAIL  */
706
707
708             if ( (Inbuf[start][off] < '0') || (Inbuf[start][off] > '9') )
709             {
710                 /* ERROR MESSAGE DOES NOT START WITH A NUMBER */
711
```

```
712             (void) strcpy(message1, &Inbuf[start][off]);
713
714             /* In some instances the next line could be a continuation    */
715             /* of the previous line like:                                 */
716             /*                                                            */
717             /*      MSG! ?D No element has been found in field            */
718                  /*      MSG!  FL.F with mdrfeat1 as its value                 */
719             /*                                                            */
720             /* So our assumptions are: if the next line starts with a     */
721             /* question mark or a number, the next line is a new message. */
722
723             if (start + 1 > stop)
724             {
725                 /******************/
726                 /* ONE LINE ERROR */
727                 /******************/
728
729                 message2[0]  = '\0';
730             }
731             else if (Inbuf[start+1][off] == '?')
732             {
733                 /******************/
734                 /* ONE LINE ERROR */
735                 /******************/
736
737                 message2[0]  = '\0';
738             }
739             else if ((Inbuf[start+1][off] >= '0') &&
740                      (Inbuf[start+1][off] <= '9') )
741             {
742                 /******************/
743                 /* ONE LINE ERROR */
744                 /******************/
745
746                 message2[0]  = '\0';
747             }
748             else
749             {
750                 /******************/
```

```
751                 /* TWO LINE ERROR */
752                 /******************/
753
754                 start++;
755                 (void) strcpy(message2, &Inbuf[start][off]);
756             }
757         }
758         else
759         {
760                 /******************/
761                 /* TWO LINE ERROR */
762                 /******************/
763
764             (void) strcpy(message1, &Inbuf[start][off]);
765             start++;
766             (void) strcpy(message2, &Inbuf[start][off]);
767         }
768         start++;
769
770         /* Add the current error message to the dictionary. */
771
772
773         pos = inserr(message1, message2);
774
775         /* Append to the index to include this error  */
776
777         if (strlen(tmpindx) == 0)
778         {
779             (void) sprintf(tmpindx, "%d", pos);
780             no_eindx = 1;
781         }
782         else
783         {
784             (void) sprintf(tmpnum, " %d", pos);
785
786             /* Will the new index fit? */
787
788             if ( (strlen(tmpindx) + strlen(tmpnum)) > ERRMSGSZ)
789             {
```

```
790                 (void) fprintf(stderr,
791                                 "\nError:  Error index buffer exceeded!\n");
792
793                 (void) fprintf(stderr, "Current Index buffer contains:\n");
794                 (void) fprintf(stderr, "[%s]\n\n", tmpindx);
795
796                 (void) fprintf(stderr, "Trying to add index [%s]\n",tmpnum);
797                 (void) fprintf(stderr, "\n");
798
799                    exit(1);
800             }
801             else
802             {
803              (void) strcat(tmpindx, tmpnum);
804                 no_eindx++;
805             }
806         }
807     }
808
809     /*****************************************************************/
810     /* The string tmpindx now contains all the indexes into the error     */
811     /* dictionary for the current error messages.  Now determine if we    */
812     /* have seen the current cluster of error messages before.            */
813     /*****************************************************************/
814
815     if (pkgflag == 1)
816     {
817         /*************************/
818         /* CURRENT PACKAGE ERRORS */
819         /*************************/
820
821         for (i=1; i<= Nperr.entries; i++)
822         {
823             if ((strcmp(Nperr.nerr[i].errindx, tmpindx)) == 0)
824             {
825                 /* We already have this error message */
826
827                 Nperr.nerr[i].no_entries++;
828                 break;
```

```
829             }
830         }
831
832         if (i > Nperr.entries)
833         {
834             /*****************************************/
835             /* New error message - add it to our list */
836             /*****************************************/
837
838             /* If you haven't opened the unique error */
839             /* message file - open it                 */
840
841             if (Nperr.entries == 0)
842             {
843                 if ((Efp = fopen(Err_file, "w")) == NULL)
844                 {
845                     (void) fprintf(stderr,
846                             "\nError: Can't open Error output file: %s\n",
847                              Err_file);
848
849                     (void) fclose(Efp);
850                     (void) fclose(Ifp);
851                     (void) fclose(Rfp);
852
853                     exit(1);
854                 }
855             }
856             else
857             {
858                 /* Add a blank line to seperate this error cluster from   */
859                 /* the previous error cluster                             */
860
861                 (void) fprintf(Efp, "%s", BLANKS);
862             }
863
864             Nperr.entries++;
865
866             if (Nperr.entries >= MAXERRMSG)
867             {
```

```
868             (void) fprintf(stderr,
869                    "\nError:  Current error message data exceeds maximum array size\n");
870             exit(1);
871         }
872
873         /*********************************/
874         /* Store the current error indexes */
875         /*********************************/
876
877         (void) strcpy(Nperr.nerr[Nperr.entries].errindx, tmpindx);
878         Nperr.nerr[Nperr.entries].no_entries = 1;
879         Nperr.nerr[Nperr.entries].no_indexes = no_eindx;
880
881         /* Loop through the tmpindx to pull out each error message out*/
882         /* of the error dictionary.  A space will seperate each index */
883
884         indxlen    = strlen(tmpindx);
885         tmpnum[0]  = '\0';
886         no_elines  = 0;
887         j          = 0;
888
889         for (i=0; i <= indxlen; i++)
890         {
891             if ((i == indxlen) || (tmpindx[i] == ' '))
892             {
893                 tmpnum[j] = '\0';
894                 indxnum   = atoi(tmpnum);
895
896                 /*****************************************************/
897                 /* Write error messages out to the UNIQUE error file */
898                 /*****************************************************/
899
900                 (void) fprintf(Efp, "%s", Errdict.msg[indxnum].em1);
901                 no_elines++;
902                 if ((strlen(Errdict.msg[indxnum].em2)) > 0)
903                 {
904                     (void) fprintf(Efp, "%s", Errdict.msg[indxnum].em2);
905                     no_elines++;
906                 }
```

```
                    j = 0;
                }
                else
                {
                   tmpnum[j] = tmpindx[i];
                   j++;
                }
            }
            Nperr.nerr[Nperr.entries].no_lines = no_elines;

        }
    }
    else
    {
        /***************************/
        /* PREVIOUS PACKAGE ERRORS */
        /***************************/

        /* The previous package error file contains only unique error     */
        /* messages.  So add to the count of error messages and insert    */
        /* the error indexes into the dictionary.                         */

        Operr.entries++;

        if (Operr.entries >= MAXERRMSG)
        {
            (void) fprintf(stderr,
                   "\nError:  Previous error message data exceeds maximum array size\n");
            exit(1);
        }

        /*******************************/
        /* Store the old error indexes */
        /*******************************/

        (void) strcpy(Operr.oerr[Operr.entries].errindx, tmpindx);
        Operr.oerr[Operr.entries].no_indexes = no_eindx;

        /* Loop through the tmpindx to pull out each error message out*/
```

```
946           /* of the error dictionary.  A space will seperate each index */
947
948           indxlen    = strlen(tmpindx);
949           tmpnum[0]  = '\0';
950           no_elines  = 0;
951           j          = 0;
952
953           for (i=0; i < indxlen; i++)
954           {
955               if ( tmpindx[i] == ' ' )
956               {
957                   tmpnum[j] = '\0';
958                   indxnum   = atoi(tmpnum);
959
960                   no_elines++;
961                   if ((strlen(Errdict.msg[indxnum].em2)) > 0)
962                   {
963                       no_elines++;
964                   }
965                   j = 0;
966               }
967               else
968               {
969                  tmpnum[j] = tmpindx[i];
970                  j++;
971               }
972           }
973           Operr.oerr[Operr.entries].no_lines = no_elines;
974
975       }
976
977       return;
978   }
979   /* EOF - bldindx */
980
981
```

```
/*****************************************************************************
 *
 * ****************
 * *** FUNCTION ***    PROGRAM UNIT  PROLOGUE:
 * ****************
 *
 * Name:               inserr()
 *
 * Abstract:           This function will insert unique error messages into
 *                     error message dictionary.
 *
 *    Programmer(s):   G. R. Daly
 *
 *          Company:   TBD
 *             Date:   Fri Oct 14 15:50:00 CST
 *
 *
 * Loadable package:   None.
 *
 * Usage:              bldindx(emsg1, emsg2)
 *
 *
 * Parameters:         emsg1      - character string error message - line 1
 *                     emsg2      - character string error message - line 2
 *                                  (could be null)
 *
 * Externals:          None.
 *
 *
 * Returns:            Return Value(s) -
 *
 *                     position withing the error dictionary where the error
 *                     exists.
 *
 *
 * Description:        This function will insert any new error messages.
 *                     It will return the position in the dictionary where
 *                     the error message exists.
 *
```

```
021     * Implementation:     Standard UNIX.
022     *
023     *
024     * Warnings:           None.
025     *
026     * Examples:           None.
027     *
028     * Calls:              strcpy()    - copies one string to another
029     *                     strcmp()    - compares one character string to another
030     *
031     * Macros:             None.
032     *
033     *
034     * See Also:           Application Usage Guide.
035     *
036     ******************************************************************************/
037
038
```

```
039  /* BOF - inserr */
040
041  int
042  inserr(emsg1, emsg2)
043  char *emsg1;                  /* error message for line 1          */
044  char *emsg2;                  /* error message for line 2          */
045  {
046       /* System Calls */
047
048       extern int   strcmp();            /* compares character strings */
049       extern char *strcpy();            /* copies character strings   */
050
051
052       /* Local Variables */
053
054       int   i;                  /* index variable            */
055
056       /* BEGIN EXECUTABLE CODE */
057
058       for (i=1; i <= Errdict.entries; i++)
059       {
060           if ((strcmp(emsg1, Errdict.msg[i].em1)) == 0)
061           {
062               /*********************************************/
063               /* ERROR MESSAGE ALREADY EXISTS IN DICTIONARY */
064               /*********************************************/
065
066               /* Don't duplicate existing message */
067
068               return(i);
069           }
070       }
071
072       /***********************************/
073       /* ADD ERROR MESSAGE TO DICTIONARY */
074       /***********************************/
075
076       Errdict.entries++;
077       (void) strcpy(Errdict.msg[Errdict.entries].em1, emsg1);
```

```
078
079       if ((strlen(emsg2)) > 0)
080       {
081           (void) strcpy(Errdict.msg[Errdict.entries].em2, emsg2);
082       }
083       else
084       {
085           Errdict.msg[Errdict.entries].em2[0] = '\0';
086       }
087
088       return(Errdict.entries);
089   }
090
091   /* EOF - inserr */
```