```
/*---------------------------------------------------------------
 * cPost.c  : c language formatter
 *---------------------------------------------------------------
 * 10-10-91 originally by Patrick J. Mueller
 * 12-03-92 converted from cBook to cPost
 * 10-13-99 added Java tokens (Steven Pothoven)
 *---------------------------------------------------------------*/

#define PROGRAM_VERS  "1.5"
#define PROGRAM_NAME  "cPost"
#define PROGRAM_YEAR  "1999"
#define PROGRAM_AUTH  "Patrick J. Mueller"
#define PROGRAM_ADDR  "(pmuellr@acm.org)"
#define PROGRAM_ENVV  "CPOST"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <signal.h>
#include <time.h>

#include "parsearg.h"

#include "ctok.h"
#define DEFINE_GLOBALS
#include "cpost.h"
#include "tokfile.h"

#include "cposthdr.h"

/*---------------------------------------------------------------
 * global variables
 *---------------------------------------------------------------*/
int  AllDone = 0;

static char *ReservedTokens[] =
  {
  /*---------------------------------------------------------------
   * data types
   *---------------------------------------------------------------*/
  "auto", "char", "const", "double", "enum", "extern", "float", "int",
  "long", "register", "short", "signed", "static", "struct", "union",
  "unsigned", "void", "volatile",

  /*---------------------------------------------------------------
   * other keywords
   *---------------------------------------------------------------*/
  "break", "case", "continue", "default", "do", "else", "for", "goto",
  "if", "return", "sizeof", "switch", "typedef", "while",

  /*---------------------------------------------------------------
   * saa c extensions
   *---------------------------------------------------------------*/
  "_Packed","_System","_Optlink", "_Far16", "_Cdecl", "_Pascal"
  };

/*---------------------------------------------------------------
 * c++ reserved words
 *---------------------------------------------------------------*/
static char *CppReservedTokens[] =
  {
  "catch", "class", "delete", "friend", "inline", "new", "operator",
  "private", "protected", "public", "template", "this", "throw", "try",
  "virtual"
  };

/*---------------------------------------------------------------
 * Java reserved words
 *---------------------------------------------------------------*/
static char *JavaReservedTokens[] =
  {
     "abstract", "boolean", "break", "byte", "case", "catch", "char",
     "class", "const", "continue", "default", "do", "double", "else",
     "extends", "final", "finally", "float", "for", "goto", "if",
     "implements", "import", "instanceof", "int", "interface",
     "long", "native", "new", "package", "private", "protected",
     "public", "return", "short", "static", "strictfp", "super",
```

```
79      "switch", "synchronized", "this", "throw", "throws",
80      "transient", "try", "void" "volatile", "while"
81    };
82
83  /*-------------------------------------------------------------------
84   * generate an error message and exit
85   *-------------------------------------------------------------------*/
86  void cPostError(
87     int   exitCode,
88     char *format,
89     ...
90     )
91    {
92    va_list vlist;
93
94    fprintf(stderr,"%s : ",PROGRAM_NAME);
95
96    va_start(vlist,format);
97    vfprintf(stderr,format,vlist);
98    va_end(vlist);
99
100   fprintf(stderr,"\n");
101
102   if (exitCode)
103      exit(exitCode);
104   }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------
 * print some help, assuming me is argv[0]
 *-----------------------------------------------------------------*/
static void Usage(void)
{
   fprintf(stderr,"%s %s by %s %s\n",PROGRAM_NAME,PROGRAM_VERS,PROGRAM_AUTH,PROGRAM_ADDR);
   fprintf(stderr,"\n");
   fprintf(stderr,"usage:\n");
   fprintf(stderr,"   %s <options> <filespec> <filespec> ...\n",PROGRAM_NAME);
   fprintf(stderr,"is used to produce a listing of C language files in PostScript\n");
   fprintf(stderr,"format.  The PostScript output is written to stdout.\n\n");
   fprintf(stderr,"where:\n");
   fprintf(stderr,"   <filespec> is a filespec matching C language files.\n\n");
   fprintf(stderr,"Valid options are:\n");
   fprintf(stderr,"   -b[+|-]             - enable/disable bracketing around braces\n");
   fprintf(stderr,"   -cext1,ext2,...     - treat files with extention ext1 and ext2 as C files\n");
   fprintf(stderr,"   -d[+|-]             - enable/disable duplex\n");
   fprintf(stderr,"   -hext1,ext2,...     - treat files with extention ext1 and ext2 as H files\n");
   fprintf(stderr,"   -ifile1;file2;...   - imbed files into output\n");
   fprintf(stderr,"   -kk1,k2,...         - treat k1, k2 as reserved (key=c++ adds c++ keywords)\n");
   fprintf(stderr,"   -kjava              - treat only java keywords as reserved\n");
   fprintf(stderr,"   -n#                 - separate line numbers from lines with # spaces\n");
   fprintf(stderr,"   -n0                 - do not generate line numbers\n");
   fprintf(stderr,"   -ofile              - output written to file (instead of stdout)\n");
   fprintf(stderr,"   -p[+|-]             - enable/disable page break at functions\n");
   fprintf(stderr,"   -rfile1;file2,...   - replace default PS procs with contents of files\n");
   fprintf(stderr,"   -snt or -stn        - sort files by name/type or type/name\n");
   fprintf(stderr,"   -t#                 - expand tabs to # columns\n");
   fprintf(stderr,"   -wf1,f2;f3,f4       - prepend files f1, f2 to output, append files f3, f4\n");
   fprintf(stderr,"   -xx,y               - coordinates for page adjustment\n");
   fprintf(stderr,"   -ypath              - path to use for temporary files\n");
   fprintf(stderr,"   -?                  - display this help\n\n");
   fprintf(stderr,"Default options are:\n");
   fprintf(stderr,"   -b+ -d- -cc -hh -n2 -p+ -stn -t4 -x0,0\n");
   fprintf(stderr,"Options may also be set in the environment variable %s.\n",PROGRAM_ENVV);

   exit(1);
}
```

```

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * add default key words to reserved hash
 *------------------------------------------------------------------*/
void InitializeReservedHash(
   Info *info,
   char *keyList
   )
   {
   int   i;
   char *part;

   /*---------------------------------------------------------------
    * create hash table
    *---------------------------------------------------------------*/
   info->reservedHash = HashCreate(sizeof(char *),
                                   30,
                                   (HashFunc *)IdentHash,
                                   (ListCompareFunc *)IdentCompare,
                                   cPostNoMem);

   if (!info->reservedHash)
      cPostError(1,"error creating reserved word hash table");

   for (i=0; i<sizeof(ReservedTokens)/sizeof(char *); i++)
      if (!HashAdd(info->reservedHash,&(ReservedTokens[i])))
         cPostError(1,"error adding reserved word '%s' to hash table",
                    ReservedTokens[i]);

   /*---------------------------------------------------------------
    * loop through the comma separated keys ...
    *---------------------------------------------------------------*/
   part = strtok(keyList,",");
   while (part)
      {
      /*------------------------------------------------------------
       * special c++ token
       *------------------------------------------------------------*/
      if (!Stricmp("c++",part))
         {
         for (i=0; i<sizeof(CppReservedTokens)/sizeof(char *); i++)
            if (!HashAdd(info->reservedHash,&(CppReservedTokens[i])))
               cPostError(1,"error adding reserved word '%s' to hash table",
                          CppReservedTokens[i]);

         }

      /*------------------------------------------------------------
       * use Java tokens
       *------------------------------------------------------------*/
      if (!Stricmp("java",part))
         {
         /*---------------------------------------------------------
          * remove default C tokens
          *---------------------------------------------------------*/
         for (i=0; i<sizeof(ReservedTokens)/sizeof(char *); i++)
            HashDelete(info->reservedHash,&(ReservedTokens[i]));

         /*---------------------------------------------------------
          * add Java tokens
          *---------------------------------------------------------*/
         for (i=0; i<sizeof(JavaReservedTokens)/sizeof(char *); i++)
            if (!HashAdd(info->reservedHash,&(JavaReservedTokens[i])))
               cPostError(1,"error adding reserved word '%s' to hash table",
                          JavaReservedTokens[i]);

         }

      /*------------------------------------------------------------
       * file name
       *------------------------------------------------------------*/
      else if (('@' == part[0]) && (1 != strlen(part)))
         {
         TokFileInfo  tfi;
         char        *key;

         part++;
```

```
227
228             tfi = TokFileOpen(part);
229
230             if (!tfi)
231                 cPostError(0,"error opening file '%s' for reading",part);
232
233             else
234               {
235               while (NULL != (part = TokFileNext(tfi)))
236                 {
237                 key = malloc(1 + strlen(part));
238                 if (!key)
239                     cPostError(1,"out of memory!!");
240
241                 strcpy(key,part);
242
243                 if (!HashAdd(info->reservedHash,&key))
244                     cPostError(0,"error adding reserved word '%s' to hash table; word ignored",
245                                key);
246                 }
247               }
248           }
249
250         /*-------------------------------------------------------------
251          * plain old token
252          *-------------------------------------------------------------*/
253         else if (!HashAdd(info->reservedHash,&part))
254           {
255           cPostError(0,"error adding reserved word '%s' to hash table; word ignored",part);
256           }
257
258         part = strtok(NULL,",");
259         }
260     }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * clean up file list
 *------------------------------------------------------------------*/
static int CleanUpFileList(
   File     *file,
   Info     *info
   )
   {
   PageEject *pe;

   if (!(info->oDebug & 1) && (file->tempName))
      remove(file->tempName);

   /*------------------------------------------------------------------
    * free up file fields
    *------------------------------------------------------------------*/
   free(file->name);
   free(file->pathName);

   if (*(file->ext))
      free(file->ext);

   if (file->tempName)
      free(file->tempName);

   /*------------------------------------------------------------------
    * free function lists
    *------------------------------------------------------------------*/
   ListDestroy(file->funcDefList);
   ListDestroy(file->funcProList);

   /*------------------------------------------------------------------
    * free page eject list
    *------------------------------------------------------------------*/
   pe = file->breakList;
   while (pe)
      {
      PageEject *next;

      next = pe->next;
      free(pe);
      pe   = next;
      }

   return 0;
   }

/*------------------------------------------------------------------
 * clean up function list
 *------------------------------------------------------------------*/
static int CleanUpFuncList(
   Function *func,
   Info     *info
   )
   {
   ListDestroy(func->callsList);
   ListDestroy(func->calledByList);

   free(func->name);

   return 0;
   }
```

```
/*---------------------------------------------------------------
 * atexit processing
 *---------------------------------------------------------------*/
static void RunAtExit(void)
  {

   if (!AllDone)
      fprintf(stderr,"%s : Program terminated.\n",PROGRAM_NAME);

   /*------------------------------------------------------------
    * erase any temporary files we might have open
    *------------------------------------------------------------*/
   if (!AllDone)
      fprintf(stderr,"%s : Cleaning up temporary files.\n",PROGRAM_NAME);

   ListIterate(info.fileList,(ListIterateFunc *)CleanUpFileList,&info);

   /*------------------------------------------------------------
    * destroy file list
    *------------------------------------------------------------*/
   ListDestroy(info.fileList);

   /*------------------------------------------------------------
    * destroy hash tables
    *------------------------------------------------------------*/
   HashDestroy(info.identHash);
   HashDestroy(info.reservedHash);

   /*------------------------------------------------------------
    * destroy function list
    *------------------------------------------------------------*/
   ListIterate(info.funcTree,(ListIterateFunc *)CleanUpFuncList,&info);
   ListDestroy(info.funcTree);

   /*------------------------------------------------------------
    * dump memory (if debug enabled
    *------------------------------------------------------------*/
#if defined(__DEBUG_ALLOC__)
   _dump_allocated(0);
#endif
  }

/*---------------------------------------------------------------
 * signal handler for program interruption
 *---------------------------------------------------------------*/
void SignalHandler(int sig)
  {
   exit(1);
  }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*---------------------------------------------------------------
 * print function name symbol defintions
 *---------------------------------------------------------------*/
static int PrintFunctionDefinition(
   Function *func,
   Info     *info
   )
  {
   fprintf(info->oFile,
           ".nameit symbol=fn%4.4d gmltype=hp%c size='+%d' text='%s'\n",
           func->id, '2', 3, func->name);

   return 0;
  }
```

```
395
396 /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
397 /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
398
399 /*---------------------------------------------------------------------
400  * compute calls and called bys, replace if > than max
401  *---------------------------------------------------------------------*/
402 static int CountFunctionRefs(
403    Function *func,
404    Info     *info
405    )
406    {
407    int count;
408
409    count  = ListCount(func->callsList);
410    count += ListCount(func->calledByList);
411
412    if (count > info->count1)
413       info->count1 = count;
414
415    return 0;
416    }
417
418 /*---------------------------------------------------------------------
419  * get maximum number of calls and called bys for all the functions
420  * so figure out which table to use
421  *---------------------------------------------------------------------*/
422 int GetMaxFuncTableEntries(
423    Info      *info
424    )
425    {
426
427    info->count1 = 0;
428
429    ListIterate(info->funcTree,
430                (ListIterateFunc *)CountFunctionRefs,
431                info);
432
433    return info->count1;
434    }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * parse command line for options
 *------------------------------------------------------------------*/
static void GetOptions(
   int    *argc,
   char   *argv[],
   Info   *info
   )
   {
   int              oHelp;
   char            *oBrack;
   char            *oTabs;
   char            *oCtype;
   char            *oHtype;
   char            *oSort;
   char            *oFile;
   char            *oSpace;
   char            *oKeys;
   char            *oBreak;
   char            *oTemp;
   char            *oImbed;
   char            *oDuplex;
   char            *oXlate;
   char            *numLeft;
   char            *s1;
   char            *s2;
   char            *oWrap;
   char            *oRepHdr;

   /*---------------------------------------------------------------
    * parse arguments
    *---------------------------------------------------------------*/
   parsearg(argc,argv,0,PROGRAM_ENVV,"-",
            Strcopy("? b@ c@ d@ h@ k@ i@ n@ o@ p@ r@ s@ t@ w@ x@ y@"),
            &oHelp,&oBrack,&oCtype,&oDuplex,&oHtype,&oKeys,&oImbed,
            &oSpace,&oFile,&oBreak,&oRepHdr,&oSort,&oTabs,&oWrap,
            &oXlate,&oTemp);

   /*---------------------------------------------------------------
    * check parms
    *---------------------------------------------------------------*/
   if (oHelp || (*argc < 2))
      Usage();

   if ('?' == *argv[1])
      Usage();

   /*---------------------------------------------------------------
    * apply option defaults
    *---------------------------------------------------------------*/
   if ((NULL == oBrack ) || ('\0' == *oBrack )) oBrack  = "+";
   if ((NULL == oCtype ) || ('\0' == *oCtype )) oCtype  = "c";
   if ((NULL == oDuplex) || ('\0' == *oDuplex)) oDuplex = "-";
   if ((NULL == oHtype ) || ('\0' == *oHtype )) oHtype  = "h";
   if ((NULL == oKeys  ) || ('\0' == *oKeys  )) oKeys   = "";
   if ((NULL == oImbed ) || ('\0' == *oImbed )) oImbed  = "";
   if ((NULL == oSpace ) || ('\0' == *oSpace )) oSpace  = "2";
   if ((NULL == oFile  ) || ('\0' == *oFile  )) oFile   = NULL;
   if ((NULL == oBreak ) || ('\0' == *oBreak )) oBreak  = "+";
   if ((NULL == oRepHdr) || ('\0' == *oRepHdr)) oRepHdr = "";
   if ((NULL == oSort  ) || ('\0' == *oSort  )) oSort   = "tn";
   if ((NULL == oTabs  ) || ('\0' == *oTabs  )) oTabs   = "4";
   if ((NULL == oWrap  ) || ('\0' == *oWrap  )) oWrap   = "";
   if ((NULL == oXlate ) || ('\0' == *oXlate )) oXlate  = Strcopy("0,0");
   if ((NULL == oTemp  ) || ('\0' == *oTemp  )) oTemp   = "";

   /*---------------------------------------------------------------
    * bracketing option
    *---------------------------------------------------------------*/
   info->oBrack = (int) strtol(oBrack,NULL,10);
   if (0 == info->oBrack)
      {
      if ((1 != strlen(oBrack)) || (NULL == strchr("-+",*oBrack)))
         cPostError(1,"invalid value on -b option");

      if ('+' == *oBrack)
         info->oBrack = 1000;
      else
```

```
517         info->oBrack = 0;
518       }
519
520     /*---------------------------------------------------------------
521      * extensions for C files
522      *---------------------------------------------------------------*/
523     info->oCtype = oCtype;
524
525     /*---------------------------------------------------------------
526      * duplex
527      *---------------------------------------------------------------*/
528     if ((1 != strlen(oDuplex)) || (NULL == strchr("-+",*oDuplex)))
529       cPostError(1,"invalid value on -d option");
530
531     info->oDuplex  = ('+' == *oDuplex);
532
533     /*---------------------------------------------------------------
534      * extensions for H files
535      *---------------------------------------------------------------*/
536     info->oHtype = oHtype;
537
538     /*---------------------------------------------------------------
539      * reserved words
540      *---------------------------------------------------------------*/
541     InitializeReservedHash(info,oKeys);
542
543     /*---------------------------------------------------------------
544      * imbed option
545      *---------------------------------------------------------------*/
546     info->oImbed  = oImbed;
547
548     /*---------------------------------------------------------------
549      * space option
550      *---------------------------------------------------------------*/
551     info->oSpace = (int) strtol(oSpace,&numLeft,10);
552     if (*numLeft || (info->oSpace < 0))
553       cPostError(1,"invalid value on -n option");
554
555     /*---------------------------------------------------------------
556      * output file option
557      *---------------------------------------------------------------*/
558     if (NULL == oFile)
559       info->oFile = stdout;
560     else
561       {
562       info->oFile = fopen(oFile,"w");
563       if (NULL == info->oFile)
564         cPostError(1,"error opening output file %s for writing",oFile);
565       }
566
567     /*---------------------------------------------------------------
568      * page break option
569      *---------------------------------------------------------------*/
570     if ((1 != strlen(oBreak)) || (NULL == strchr("-+",*oBreak)))
571       cPostError(1,"invalid value on -p option");
572
573     info->oBreak  = ('+' == *oBreak);
574
575     /*---------------------------------------------------------------
576      * replace PS header
577      *---------------------------------------------------------------*/
578     info->oRepHdr = oRepHdr;
579
580     /*---------------------------------------------------------------
581      * sort option
582      *---------------------------------------------------------------*/
583     if ((0 != Stricmp("nt",oSort)) && (0 != Stricmp("tn",oSort)))
584       cPostError(1,"invalid value on -s option");
585
586     info->oSort  = Strupr(oSort);
587
588     /*---------------------------------------------------------------
589      * tabs option
590      *---------------------------------------------------------------*/
591     info->oTabs = (int) strtol(oTabs,NULL,10);
592     if (0 == info->oTabs)
593       cPostError(1,"invalid value on -t option");
594
595     /*---------------------------------------------------------------
596      * wrap PS around output
```

```
597     *-------------------------------------------------------------*/
598     info->oWrapB = strtok(oWrap,";");
599     info->oWrapA = strtok(NULL,"");
600
601     /*-------------------------------------------------------------
602      * translate option
603      *-------------------------------------------------------------*/
604     s1 = strtok(oXlate,",");
605     s2 = strtok(NULL,"");
606
607     if (!s1 || !s2)
608        cPostError(1,"invalid value on -x option");
609
610     info->oXlateX = (int) strtol(s1,NULL,10);
611     info->oXlateY = (int) strtol(s2,NULL,10);
612
613     /*-------------------------------------------------------------
614      * temp path
615      *-------------------------------------------------------------*/
616     if (!strlen(oTemp))
617        info->oTemp = "";
618
619     else
620       {
621       char c;
622
623       c = oTemp[strlen(oTemp) - 1];
624       if (('\\' == c) || ('/' == c))
625          info->oTemp = oTemp;
626       else
627         {
628         info->oTemp = malloc(2+strlen(oTemp));
629         strcpy(info->oTemp,oTemp);
630         strcat(info->oTemp,"/");
631         }
632       }
633
634
635   }
```

```
636
637  /*-------------------------------------------------------------------
638   * copy one file stream to another
639   *-------------------------------------------------------------------*/
640  void copyFile(
641     FILE *fileFrom,
642     FILE *fileTo
643     )
644    {
645  #define BUFFER_SIZE 8192
646     char *buffer;
647
648     /*-------------------------------------------------------------------
649      * allocate buffer
650      *-------------------------------------------------------------------*/
651     buffer = malloc(BUFFER_SIZE);
652     if (!buffer)
653        cPostError(1,"out of memory!!");
654
655     /*-------------------------------------------------------------------
656      * copy file buffer at a time
657      *-------------------------------------------------------------------*/
658     while (!feof(fileFrom))
659       {
660       int count;
661       count = fread(buffer,1,BUFFER_SIZE,fileFrom);
662       fwrite(buffer,1,count,fileTo);
663       }
664
665     /*-------------------------------------------------------------------
666      * free the buffer
667      *-------------------------------------------------------------------*/
668     free(buffer);
669     }
```

```
670
671 /*-------------------------------------------------------------
672  * process the imbed file option
673  *-------------------------------------------------------------*/
674 void processImbedFile(
675    char *imbedFileName
676    )
677  {
678  FILE *file;
679
680  /*-------------------------------------------------------------
681   * while we have imbedFileNames
682   *-------------------------------------------------------------*/
683  imbedFileName = strtok(imbedFileName,";,");
684  while (imbedFileName)
685    {
686    /*-------------------------------------------------------------
687     * open the imbed file
688     *-------------------------------------------------------------*/
689    file = fopen(imbedFileName,"r");
690
691    /*-------------------------------------------------------------
692     * print error if not found, or copy it in if found
693     *-------------------------------------------------------------*/
694    if (!file)
695       cPostError(0,"unable to open file '%s' for reading",imbedFileName);
696    else
697      {
698      copyFile(file,info.oFile);
699      fclose(file);
700      }
701
702    /*-------------------------------------------------------------
703     * get next imbed file name
704     *-------------------------------------------------------------*/
705    imbedFileName = strtok(NULL,";,");
706    }
707  }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * main program
 *------------------------------------------------------------------*/
int main(
   int   argc,
   char *argv[]
   )
   {
   int         i;
   char        dateStr[30];
   struct tm *tm;
   time_t      t;
   char       *origParms;

   /*------------------------------------------------------------------
    * check for help
    *------------------------------------------------------------------*/
   if ((1 == argc) || ('?' == *(argv[1])))
      Usage();

   /*------------------------------------------------------------------
    * get original parms
    *------------------------------------------------------------------*/
   origParms  = malloc(1);
   *origParms = 0;

   for (i=0; i<argc; i++)
      {
      origParms = realloc(origParms,2 + strlen(origParms) + strlen(argv[i]));
      strcat(origParms," ");
      strcat(origParms,argv[i]);
      }

   /*------------------------------------------------------------------
    * zero out info
    *------------------------------------------------------------------*/
   memset(&info,0,sizeof(info));

   /*------------------------------------------------------------------
    * get options
    *------------------------------------------------------------------*/
   GetOptions(&argc,argv,&info);

   /*------------------------------------------------------------------
    * buffer output
    *------------------------------------------------------------------*/
/* setvbuf(info.oFile,NULL,_IOFBF,32000); */

   /*------------------------------------------------------------------
    * put filenames in a list
    *------------------------------------------------------------------*/
   info.fileList = ListCreate(sizeof(File),
                              (ListCompareFunc *)FileNameCompare,
                              cPostNoMem);
   if (!info.fileList)
      cPostError(1,"error creating list of files");

   for (i=1; i<argc; i++)
      FileSpecAdd(&info,info.fileList,argv[i]);

   /*------------------------------------------------------------------
    * check for no files to process
    *------------------------------------------------------------------*/
   if (!ListCount(info.fileList))
      cPostError(1,"no files to process");

   /*------------------------------------------------------------------
    * intialize rest of info structure
    *------------------------------------------------------------------*/
   info.funcTree  = ListCreate(sizeof(Function),
                               (ListCompareFunc *)FunctionNameCompare,
                               cPostNoMem);
   if (!info.fileList)
      cPostError(1,"error creating list of functions");

```

```
787     info.identHash   = HashCreate(sizeof(char *),
788                                   1000,
789                                   (HashFunc *)IdentHash,
790                                   (ListCompareFunc *)IdentCompare,
791                                   cPostNoMem);
792
793
794     if (!info.identHash)
795        cPostError(1,"error creating global hash table");
796
797     /*-----------------------------------------------------------------
798      * setup error termination processing
799      *-----------------------------------------------------------------*/
800     atexit(RunAtExit);
801     signal(SIGINT,  SignalHandler);
802     signal(SIGTERM, SignalHandler);
803
804  #if defined(OPSYS_OS2) || defined(OPSYS_OS2V2)
805     signal(SIGBREAK,SignalHandler);
806  #endif
807
808     /*-----------------------------------------------------------------
809      * print header
810      *-----------------------------------------------------------------*/
811     fprintf(info.oFile,"%%! PostScript file generated by %s %s\n\n",
812             PROGRAM_NAME,PROGRAM_VERS);
813
814  /*-------------------------------------------------------------------
815   * a macro to write a line to the output file
816   *-------------------------------------------------------------------*/
817  #define p(x) fprintf(info.oFile,"%s\n",x);
818
819
820     /*-----------------------------------------------------------------
821      * write command line and environment variable setting
822      *-----------------------------------------------------------------*/
823  #if defined(ECHO_COMMAND_LINE)
824     {
825     p("%%-----------------------------------------------------------------")
826
827     fprintf(info.oFile,"%%%% this file created with the command:\n");
828     fprintf(info.oFile,"%%%%   %s\n",origParms);
829     fprintf(info.oFile,"%%%% the CPOST environment variable ");
830     if (!getenv(PROGRAM_ENVV))
831        fprintf(info.oFile,"is not set.\n");
832     else
833        {
834        fprintf(info.oFile,"is set to:\n");
835        fprintf(info.oFile,"%%%%   %s\n",getenv(PROGRAM_ENVV));
836        }
837
838     p("%%-----------------------------------------------------------------")
839     p("");
840     }
841  #endif
842
843     /*-----------------------------------------------------------------
844      * write wrapper prefix
845      *-----------------------------------------------------------------*/
846     if (info.oWrapB && strlen(info.oWrapB))
847        processImbedFile(info.oWrapB);
848
849     /*-----------------------------------------------------------------
850      * get the time
851      *-----------------------------------------------------------------*/
852     t  = time(NULL);
853     tm = localtime(&t);
854     strftime(dateStr,sizeof(dateStr)-1,"%m/%d/%y   %H:%M:%S",tm);
855
856     p("%%-----------------------------------------------------------------")
857     p("%% runtime options and values")
858     p("%%-----------------------------------------------------------------")
859     p("")
```

```
860     fprintf(info.oFile,"/printDate (%s) def\n",dateStr);
861     fprintf(info.oFile,"/oSpace %d def\n",info.oSpace);
862     fprintf(info.oFile,"/oXlate { %d %d translate } def\n",info.oXlateX,info.oXlateY);
863     fprintf(info.oFile,"/oDuplex 1 %d eq def\n",info.oDuplex);
864     fprintf(info.oFile,"/oNumber 0 %d ne def\n",info.oSpace);
865     p("")
866
867     /*-------------------------------------------------------------
868      * write replaced header ...
869      *-------------------------------------------------------------*/
870     if (info.oRepHdr && strlen(info.oRepHdr))
871       {
872       processImbedFile(info.oImbed);
873       p("");
874       processImbedFile(info.oRepHdr);
875       p("");
876       }
877
878     /*-------------------------------------------------------------
879      * or default stuff
880      *-------------------------------------------------------------*/
881     else
882       {
883       for (i=0; i< sizeof(Header_1)/sizeof(char *); i++)
884           p(Header_1[i]);
885
886       p("");
887       processImbedFile(info.oImbed);
888       p("");
889
890       for (i=0; i< sizeof(Header_2)/sizeof(char *); i++)
891           p(Header_2[i]);
892
893       p("");
894       }
895
896     /*-------------------------------------------------------------
897      * read the files. make copies
898      *-------------------------------------------------------------*/
899     fprintf(stderr,"Pass 1\n");
900     ListIterate(info.fileList,(ListIterateFunc *)Pass1,&info);
901
902     /*-------------------------------------------------------------
903      * read the copies. write the output file
904      *-------------------------------------------------------------*/
905     fprintf(stderr,"Pass 2\n");
906     ListIterate(info.fileList,(ListIterateFunc *)Pass2,&info);
907
908     /*-------------------------------------------------------------*
909      * print trailing line feed
910      *-------------------------------------------------------------*/
911     fprintf(info.oFile,"\n");
912
913     /*-------------------------------------------------------------
914      * write wrapper suffix
915      *-------------------------------------------------------------*/
916     if (info.oWrapA && strlen(info.oWrapA))
917        processImbedFile(info.oWrapA);
918
919     /*-------------------------------------------------------------
920      * close file (another line feed for luck!)
921      *-------------------------------------------------------------*/
922     fprintf(info.oFile,"\n");
923     fclose(info.oFile);
924
925     AllDone = 1;
926     return 0;
927     }
```

```
/*-------------------------------------------------------------------
 * cpostp1.c : Pass 1 of cPost
 *-------------------------------------------------------------------
 * 12-02-91 originally by Patrick J. Mueller
 * 12-03-92 converted from cBook to cPost
 *-------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ctok.h"
#include "cpost.h"
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-------------------------------------------------------------------
 * add bracketing starts in front of braces
 *-------------------------------------------------------------------*/
static void AddLBracket(
   File *file,
   char *line,
   char *mask,
   int  *maxMask,
   Tok   *tok
   )
   {
   int  lIndent;
   int  rIndent;
   int  place;
   Tok *sib;

   /*-------------------------------------------------------------
    * see if matching bracket is on this line - if so, skip
    *-------------------------------------------------------------*/
   sib = tok->sib;
   if (sib && (sib->tok.line == tok->tok.line))
      return;

   /*-------------------------------------------------------------
    * find minimum indent of left and right brace
    *-------------------------------------------------------------*/
   lIndent = strspn(line," ");
   if (!tok->sib)
      place = lIndent;
   else
      {
      rIndent = strspn(file->line[sib->tok.line-1]," ");
      place = min(lIndent,rIndent);
      }

   /*-------------------------------------------------------------
    * if no indentation on { or }, don't bracket!
    *-------------------------------------------------------------*/
   if (!place)
      return;

   /*-------------------------------------------------------------
    * we'll bracket the column BEFORE this
    *-------------------------------------------------------------*/
   place--;

   /*-------------------------------------------------------------
    * set the mask, maxMask, and write bracket to line
    *-------------------------------------------------------------*/
   mask[place]++;
   *maxMask = max(place,*maxMask);

   if (' ' == line[place])
      line[place] = '\x01';
   }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------
 * add bracketing ends
 *-----------------------------------------------------------------*/
static void AddRBracket(
   File *file,
   char *line,
   char *mask,
   int  *maxMask,
   Tok   *tok
   )
   {
   Tok *sib;

   /*-------------------------------------------------------------
    * see if matching bracket is on this line - if so, skip
    *-------------------------------------------------------------*/
   sib = tok->sib;
   if (sib && (sib->tok.line == tok->tok.line))
      return;

   /*-------------------------------------------------------------
    * safety valve
    *-------------------------------------------------------------*/
   if (-1 == *maxMask)
      return;

   /*-------------------------------------------------------
    * add end bracket to line
    *-------------------------------------------------------*/
   if (' ' == line[*maxMask])
      line[*maxMask] = '\x03';

   /*-------------------------------------------------------
    * reset mask and maxMask
    *-------------------------------------------------------*/
   mask[*maxMask]--;
   if (!mask[*maxMask])
      {
      *maxMask -= 1;
      while (-1 != *maxMask)
         if (mask[*maxMask])
            break;
         else
            *maxMask -= 1;
      }
   }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-------------------------------------------------------------------
 * add bracketing chars to line based on mask
 *-------------------------------------------------------------------*/
static char *AddMBrackets(
   File *file,
   char *line,
   char *mask,
   int   maxMask
   )
   {
   int   i;

   /*-------------------------------------------------------------
    * see if we need to make the line longer
    *-------------------------------------------------------------*/
   if (maxMask + 1 >= (int) strlen(line))
      {
      char *newLine;

      newLine = malloc(maxMask+3);
      if (!newLine)
         cPostError(1,"out of memory!!!");

      if ('\n' == line[strlen(line)-1])
         line[strlen(line)-1] = ' ';

      memset(newLine,' ',maxMask+1);
      newLine[maxMask+1] = '\n';
      newLine[maxMask+2] = 0;
      memcpy(newLine,line,strlen(line));

      free(line);
      line = newLine;
      }

   /*----------------------------------------------------------------
    * get number of first non-blank column
    *----------------------------------------------------------------*/
   maxMask = min(maxMask,(int)strspn(line," "));

   /*----------------------------------------------------------------
    * for each non-zero entry in mask, write bracket
    *----------------------------------------------------------------*/
   for (i=0; i<=maxMask; i++)
      {
      if (mask[i])
         if (' ' == line[i])
            line[i] = '\02';
      }

   return line;
   }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * strip trailing blanks off a line (that has a \n at the end!!)
 *------------------------------------------------------------------*/
static void StripTrailingBlanks(
   char *line
   )
   {
   int slen;

   slen = strlen(line);
   if ('\n' != line[slen-1])
      {
      fprintf(stderr,"line found without carriage return!!\n");
      return;
      }

   line[slen-1] = ' ';

   while (*line && (' ' == line[slen-1]))
      {
      line[slen-1] = '\0';
      slen--;
      }

   line[slen] = '\n';
   }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * add function information to trees and lists
 *------------------------------------------------------------------*/
static void AddFunctionPrototype(
   Info           *info,
   File           *file,
   char          *name
   )
   {
   Function *func;

   func = GetFunction(info,name);

   if (!ListFind(file->funcProList,&func))
      if (!ListAdd(file->funcProList,&func))
         cPostError(1,"error adding function prototype to list");
   }
```

```
231
232 /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
233 /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
234
235 /*---------------------------------------------------------------------
236  * add function information to trees and lists
237  *---------------------------------------------------------------------*/
238 static void AddFunctionDefinition(
239    Info          *info,
240    File          *file,
241    char          *name,
242    unsigned long  lineNo
243    )
244 {
245 Function *func;
246
247 func = GetFunction(info,name);
248 if (!ListFind(file->funcDefList,&func))
249    if (!ListAdd(file->funcDefList,&func))
250       cPostError(1,"error adding function definition to list");
251
252 func->fileName = file->name;
253 func->lineNo   = lineNo;
254 }
255
256 /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
257 /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
258
259 /*---------------------------------------------------------------------
260  * add function information to trees and lists
261  *---------------------------------------------------------------------*/
262 static void AddFunctionUsage(
263    Info          *info,
264    char          *calleeName,
265    char          *callerName
266    )
267 {
268 Function *caller;
269 Function *callee;
270
271 callee = GetFunction(info,calleeName);
272 caller = GetFunction(info,callerName);
273
274 if (!ListFind(caller->callsList,   &callee))
275    if (!ListAdd(caller->callsList,   &callee))
276       cPostError(1,"error adding function callee to list");
277
278 if (!ListFind(callee->calledByList,&caller))
279    if (!ListAdd(callee->calledByList,&caller))
280       cPostError(1,"error adding function caller to list");
281 }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------
 *
 *-----------------------------------------------------------------*/
PageEject *addPageEject(
   File      *file,
   Tok       *tok,
   PageEject *pageJ
   )
   {
   PageEject     *new;
   unsigned long  lineNo;
   char          *line;

   /*-------------------------------------------------------------
    * get space for new node
    *-------------------------------------------------------------*/
   new = malloc(sizeof(PageEject));
   if (!new)
      cPostError(1,"out of memory!!!");

   /*-------------------------------------------------------------
    * link into list
    *-------------------------------------------------------------*/
   if (!pageJ)
      file->breakList = new;
   else
      pageJ->next      = new;

   new->next   = NULL;
   new->lineNo = file->lines;

   /*-------------------------------------------------------------
    * find first non blank line after token
    *-------------------------------------------------------------*/
   lineNo = tok->tok.line;
   if (lineNo >= file->lines)
      return new;

   line = file->line[lineNo];
   while (strspn(line," \n") == strlen(line))
      {
      lineNo++;
      if (lineNo >= file->lines)
         return new;

      line = file->line[lineNo];
      }

   new->lineNo = lineNo;
   return new;
   }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*---------------------------------------------------------------------
 * pass 1 of cPost
 *  - expand tabs, put bracketing around braces, write output files
 *---------------------------------------------------------------------*/
int Pass1(
   File     *file,
   Info     *info
   )
   {
   Tok            *next;
   Tok            *def;
   char          *mask;
   char          *this;
   int            lineNo;
   int            i;
   FILE           *tempFile;
   Tok            *stack;
   Tok            *temp;
   int            maxMask;
   int            brackLvl;
   Tok            *lastSemi;
   int            doLastSemi;
   int            inFunc;
   PageEject      *pageJ;

   /*------------------------------------------------------------------
    * print status
    *------------------------------------------------------------------*/
   fprintf(stderr,"   Reading file %s\n",file->name);

   /*------------------------------------------------------------------
    * parse the file
    *------------------------------------------------------------------*/
   info->indent1 = 1;

   cParse(file,info);

   info->indent1 = 0;

   /*------------------------------------------------------------------
    * go down list, adding function information to function tree,
    * and computing bracket information
    *------------------------------------------------------------------*/
   next       = file->tokList;
   def        = NULL;
   stack      = NULL;
   lastSemi   = NULL;
   pageJ      = NULL;
   brackLvl   = 0;
   inFunc     = 0;
   doLastSemi = 1;

   while (next)
      {
      switch(next->extType)
         {
         /*------------------------------------------------------------
          * function stuff
          *------------------------------------------------------------*/
         case TOKEN_FUNPRO:
            AddFunctionPrototype(info,file,next->str);
            break;

         case TOKEN_FUNDEF:
            def = next;
            AddFunctionDefinition(info,file,next->str,next->tok.line);
            inFunc = 1;

            if (doLastSemi && lastSemi)
               {
               pageJ = addPageEject(file,lastSemi,pageJ);

               lastSemi   = NULL;
               doLastSemi = 0;
               }

            break;

         case TOKEN_FUNUSE:
```

```
420             AddFunctionUsage(info,next->str,def ? def->str : "");
421             break;
422
423          /*-----------------------------------------------------
424           * semicolons for page breaks
425           *-----------------------------------------------------*/
426          case TOKEN_SCOLON:
427          case TOKEN_PREPROC:
428             if (doLastSemi)
429                lastSemi = next;
430             break;
431
432          /*-----------------------------------------------------
433           * bracket stuff
434           *-----------------------------------------------------*/
435          case TOKEN_LBRACE:
436             brackLvl++;
437             next->sib    = NULL;
438             next->flags = 0;
439
440             /*-------------------------------------------------
441              * put { on the stack
442              *-------------------------------------------------*/
443             next->sib    = stack;
444             stack        = next;
445             next->flags = 1;
446
447             break;
448
449          case TOKEN_RBRACE:
450             next->sib    = NULL;
451             next->flags = 0;
452
453                next->extType = TOKEN_RBRACE;
454
455             /*-------------------------------------------------
456              * remove { from stack, point { to } and } to {
457              *-------------------------------------------------*/
458             if (!stack)
459                fprintf(stderr,"unmatched } on line %ld\n",next->tok.line);
460             else
461                {
462                temp = stack;
463                stack = stack->sib;
464
465                temp->sib    = next;
466                next->sib    = temp;
467                next->flags = 1;
468                }
469
470             if (brackLvl > 0)
471                brackLvl--;
472
473             /*-------------------------------------------------
474              * add conditional break, if it's time
475              *-------------------------------------------------*/
476             if (inFunc && !brackLvl)
477                {
478                pageJ = addPageEject(file,next,pageJ);
479
480                doLastSemi = 0;
481                }
482
483             break;
484          }
485
486       next = next->next;
487       }
488
489    /*---------------------------------------------------------
490     * check for extra { on the stack
491     *---------------------------------------------------------*/
492    while (stack)
493       {
494       fprintf(stderr,"unmatched { on line %ld\n",stack->tok.line);
495       stack = stack->sib;
496       }
497
498    /*---------------------------------------------------------
499     * now, let's add the bracketing characters
500     *---------------------------------------------------------*/
501
502    /*---------------------------------------------------------
```

```
503     * initialize mask
504     *-------------------------------------------------------------*/
505    mask = malloc(file->maxLineLen+1);
506    if (!mask)
507       cPostError(1,"out of memory!!!");
508
509    memset(mask,0,file->maxLineLen+1);
510
511    /*-------------------------------------------------------------
512     * process each line
513     *-------------------------------------------------------------*/
514    next    = file->tokList;
515    maxMask = -1;
516    for (lineNo=0; lineNo < (int)file->lines; lineNo++)
517       {
518       this = file->line[lineNo];
519
520       /*-------------------------------------------------------
521        * go to next token in this line
522        *-------------------------------------------------------*/
523       while (next && (next->tok.line <  (unsigned long)(lineNo + 1)))
524          next = next->next;
525
526       /*-------------------------------------------------------
527        * add beginning and ending brackets
528        *-------------------------------------------------------*/
529       if (info->oBrack)
530          {
531          while (next && (next->tok.line == (unsigned long)(lineNo + 1)))
532             {
533             if      ((TOKEN_LBRACE == next->extType) && (next->flags))
534                AddLBracket(file,this,mask,&maxMask,next);
535
536             else if ((TOKEN_RBRACE == next->extType) && (next->flags))
537                AddRBracket(file,this,mask,&maxMask,next);
538
539             next = next->next;
540             }
541          }
542
543       /*-------------------------------------------------------
544        * add middle brackets
545        *-------------------------------------------------------*/
546       file->line[lineNo] = AddMBrackets(file,this,mask,maxMask);
547       }
548
549    /*-------------------------------------------------------------
550     * generate temp file name
551     *-------------------------------------------------------------*/
552    file->tempName = TempFileName(&tempFile,info);
553
554    /*-------------------------------------------------------------
555     * write file to temp file
556     *-------------------------------------------------------------*/
557    for (i=0; i<(int)file->lines; i++)
558       fputs(file->line[i],tempFile);
559
560    fclose(tempFile);
561
562    /*-------------------------------------------------------------
563     * let's clean up all the memory we used
564     *-------------------------------------------------------------*/
565    free(mask);
566
567    /*-------------------------------------------------------------
568     * clean out our data structures
569     *-------------------------------------------------------------*/
570    cParseDone(file,info);
571
572    return 0;
573    }
```

```
/*-------------------------------------------------------------------
 * cpostp2.c : pass 2 of cPost
 *-------------------------------------------------------------------
 * 12-02-91 originally by Patrick J. Mueller
 * 12-03-92 converted from cBook to cPost
 *-------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ctok.h"
#include "cpost.h"

/*-------------------------------------------------------------------
 * static buffer
 *-------------------------------------------------------------------*/
#define BUFFER_LEN  8000
#define BRACKET_LEN  500

static unsigned char Buffer    [BUFFER_LEN];
static unsigned char FuncBuff  [MAX_IDENT_LEN+1];
static unsigned char CurrFunc  [MAX_IDENT_LEN+1];
static unsigned char BrackInfo [BRACKET_LEN];

/*-------------------------------------------------------------------
 * global variables
 *-------------------------------------------------------------------*/
static FILE           *oFile;
static PageEject      *pageJ;
static char           *Xchars[256];
static unsigned char   currFont;
static unsigned long   lineNo;
static int             col;
```

```

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*--------------------------------------------------------------------
 * write out the buffer, translating as we go
 *--------------------------------------------------------------------*/
static void WriteOut(
   unsigned char     *buffer,
   int                 len
   )
   {
   int           i;
   char         *xlate;

   /*-----------------------------------------------------------------
    * print each character in buffer
    *-----------------------------------------------------------------*/
   for (i=0; i<len; i++)
      {
      xlate = Xchars[buffer[i]];

      /*--------------------------------------------------------------
       * check for end of line
       *--------------------------------------------------------------*/
      if ('\n' == buffer[i])
         {
         lineNo++;

         /*-----------------------------------------------------------
          * print end of line info
          *-----------------------------------------------------------*/
         fprintf(oFile,")] showLine %s\n",BrackInfo);
         BrackInfo[0] = 0;

         /*-----------------------------------------------------------
          * see if there is a new function definition
          *-----------------------------------------------------------*/
         if (*CurrFunc)
            {
            fprintf(oFile,"/currFunc (%s) def\n",CurrFunc);
            *CurrFunc = 0;
            }

         /*-----------------------------------------------------------
          * write page eject if we need to
          *-----------------------------------------------------------*/
         if (info.oBreak && pageJ && (pageJ->lineNo == lineNo))
            {
            if (pageJ->next)
               {
               int lines;

               lines = pageJ->next->lineNo - pageJ->lineNo;
               fprintf(oFile,"%d linesFit\n",lines);

               pageJ = pageJ->next;
               }
            }

         /*-----------------------------------------------------------
          * print start of line info
          *-----------------------------------------------------------*/
         fprintf(oFile,"[%c (",currFont);

         col = 0;
         }

      /*--------------------------------------------------------------
       * plain old character
       *--------------------------------------------------------------*/
      else if (NULL == xlate)
         {
         col++;

/*       fwrite(&(buffer[i]),1,1,oFile); */
         fputc(buffer[i],oFile);
         }

      /*--------------------------------------------------------------
       * character to translate
       *--------------------------------------------------------------*/
      else
```

```
118             {
119             char *bChar;
120             char  nBuff[20];
121
122             col++;
123
124             /*-------------------------------------------------------
125              * check for bracket character
126              *-------------------------------------------------------*/
127             bChar = NULL;
128             switch(buffer[i])
129               {
130               case '\x01' : bChar = "u"; break;
131               case '\x02' : bChar = "m"; break;
132               case '\x03' : bChar = "l"; break;
133               }
134
135             if (bChar && (strlen(BrackInfo) < BRACKET_LEN - 30))
136               {
137               sprintf(nBuff,"%d",col);
138
139
140               strcat(BrackInfo," ");
141               strcat(BrackInfo,nBuff);
142               strcat(BrackInfo," ");
143               strcat(BrackInfo,bChar);
144               strcat(BrackInfo,"B");
145               }
146
147             /*-------------------------------------------------------
148              * print translation
149              *-------------------------------------------------------*/
150 /*          fwrite(xlate,1,strlen(xlate),oFile); */
151             fputs(xlate,oFile);
152             }
153         }
154     }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * pass 2 - add formatting tags
 *------------------------------------------------------------------*/
int Pass2(
   File     *file,
   Info     *info
   )
   {
   Tok            *next;
   unsigned long  offs;
   unsigned long  len;
   FILE           *iFile;
   int            i;

   /*------------------------------------------------------------------
    * initialize some stuff
    *------------------------------------------------------------------*/
   oFile = info->oFile;

   for (i=0; i<256; i++)
      {
      Xchars[i] = NULL;
      }

   Xchars['(']  = "\\(";     /* ( -> \( */
   Xchars[')']  = "\\)";     /* ) -> \) */
   Xchars['\\'] = "\\\\";    /* \ -> \\ */

   Xchars['\x01'] = " ";
   Xchars['\x02'] = " ";
   Xchars['\x03'] = " ";

   /*------------------------------------------------------------------
    * print status and read file
    *------------------------------------------------------------------*/
   fprintf(stderr,"   Reading file %s\n",file->name);
   cParse(file,info);

   /*------------------------------------------------------------------
    * write header
    *------------------------------------------------------------------*/
   fprintf(info->oFile,"\n%%---------------------------------------------\n");

   if (file->date && file->time && *file->date && *file->time)
      fprintf(info->oFile,"(%s) (%s   %s) startFile\n",file->name,
                          file->date,file->time);
   else
      fprintf(info->oFile,"(%s) () startFile\n",file->name);

   /*------------------------------------------------------------------
    * first line processing
    *------------------------------------------------------------------*/
   CurrFunc[0]   = 0;
   BrackInfo[0] = 0;
   fprintf(oFile,"[n (");

   /*------------------------------------------------------------------
    * open the input file
    *------------------------------------------------------------------*/
   if (file->tempName)
      iFile = fopen(file->tempName,"r");
   else
      iFile = fopen(file->pathName,"r");

   if (!iFile)
      cPostError(1,"error opening file for reading");

   /*------------------------------------------------------------------
    * initialize token processing
    *------------------------------------------------------------------*/
   next     = file->tokList;
   pageJ    = file->breakList;
   offs     = 0;
   lineNo   = 0;
   col      = 0;
   currFont = 'n';

```

```
236  /*-------------------------------------------------------------
237   * loop through tokens
238   *-------------------------------------------------------------*/
239  while (next)
240    {
241    /*---------------------------------------------------------
242     * read boring stuff up to first token
243     *---------------------------------------------------------*/
244    if (next->tok.offs != offs)
245      {
246      /*---------------------------------------------------------
247       * read and write it
248       *---------------------------------------------------------*/
249      len = next->tok.offs - offs;
250      while (len > BUFFER_LEN)
251        {
252        fread(Buffer,1,BUFFER_LEN,iFile);
253        WriteOut(Buffer,BUFFER_LEN);
254        len -= BUFFER_LEN;
255        }
256
257      fread(Buffer,1,(int)len,iFile);
258      WriteOut(Buffer,(int)len);
259
260      /*---------------------------------------------------------
261       * update the pointer
262       *---------------------------------------------------------*/
263      offs = next->tok.offs;
264      }
265
266    /*---------------------------------------------------------
267     * write the token
268     *---------------------------------------------------------*/
269    switch (next->extType)
270      {
271      /*---------------------------------------------------------
272       * get it's font character
273       *---------------------------------------------------------*/
274      case TOKEN_RESER     : currFont = 'k'; break;
275      case TOKEN_PREPROC   : currFont = 'p'; break;
276      case TOKEN_COMMENT   : currFont = 'c'; break;
277      case TOKEN_IDENT     : currFont = 'i'; break;
278      case TOKEN_FUNDEF    : currFont = 'd'; break;
279      case TOKEN_FUNPRO    : currFont = 'f'; break;
280      case TOKEN_FUNUSE    : currFont = 'f'; break;
281      default :              currFont = 'n'; break;
282      }
283
284    /*---------------------------------------------------------
285     * copy function name into buffer for function definitions
286     *---------------------------------------------------------*/
287    if (TOKEN_FUNDEF == next->extType)
288      strcpy(CurrFunc,next->str);
289
290    /*---------------------------------------------------------
291     * read and write token
292     *---------------------------------------------------------*/
293    if (currFont != 'n')
294      fprintf(oFile,") %c (",currFont);
295
296    len = next->tok.len;
297    while (len > BUFFER_LEN)
298      {
299      fread(Buffer,1,(int)BUFFER_LEN,iFile);
300      WriteOut(Buffer,(int)BUFFER_LEN);
301      len -= BUFFER_LEN;
302      }
303
304    fread(Buffer,1,(int)len,iFile);
305    WriteOut(Buffer,(int)len);
306
307    if (currFont != 'n')
308      fprintf(info->oFile,") n (");
309
310    currFont = 'n';
311
312    /*---------------------------------------------------------
313     * prepare for next token
314     *---------------------------------------------------------*/
315    offs += next->tok.len;
316    next = next->next;
```

```
317     }
318
319     /*-------------------------------------------------------------
320      * last line processing
321      *-------------------------------------------------------------*/
322     fprintf(oFile,")] showLine %s\nendFile",BrackInfo);
323
324     /*-------------------------------------------------------------
325      * close file
326      *-------------------------------------------------------------*/
327     fclose(iFile);
328
329     /*-------------------------------------------------------------
330      * free parser storage
331      *-------------------------------------------------------------*/
332     cParseDone(file,info);
333
334     return 0;
335 }
```

```c
/*-----------------------------------------------------------------
 * cPostPar.c : higher level parser for cPost
 *-----------------------------------------------------------------
 * 03-19-92 originally by Patrick J. Mueller
 * 12-03-92 converted from cBook to cPost
 *-----------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ctok.h"
#include "cpost.h"

/*-----------------------------------------------------------------
 * is string a C keyword
 *-----------------------------------------------------------------*/
static int IsKeyword(
    Info *info,
    char *str
    )
    {
    if (HashFind(info->reservedHash,&str))
        return 1;
    else
        return 0;
    }
```

```

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * add function information to tokens in list
 *------------------------------------------------------------------*/
static void AddFunctionInfoToTokens(
   File *file
   )
   {
   Tok *next;
   Tok *func;

   /*---------------------------------------------------------------
    * loop through the file
    *---------------------------------------------------------------*/
   next = file->tokList;
   while (NULL != next)
      {
      /*------------------------------------------------------------
       * continue reading till we have an identifier
       *------------------------------------------------------------*/
      if (TOKEN_IDENT != next->extType)
         {
         next = next->next;
         continue;
         }

      /*---------------------------------------------------------
       * if next token is (, this is a function name!
       *---------------------------------------------------------*/
      func = next;
      next = next->next;
      if (NULL == next)
         return;

      while ((TOKEN_COMMENT == next->extType) ||
             (TOKEN_PREPROC == next->extType))
         {
         next = next->next;

         if (NULL == next)
            return;
         }

      if (TOKEN_LPAREN == next->extType)
         {

         /*------------------------------------------------------
          * if we're in braces, it's usage
          *------------------------------------------------------*/
         if (next->nestBrace > 0)
            func->extType = TOKEN_FUNUSE;

         /*------------------------------------------------------
          * otherwise it's a prototype or definition
          *------------------------------------------------------*/
         else
            {
            /*---------------------------------------------------
             * get to the top level parenthesis
             *---------------------------------------------------*/
            while (next->nestParen > 0)
               {
               next = next->next;
               if (NULL == next)
                  return;
               }

            /*---------------------------------------------------
             * if next token is ;, it's a prototype, otherwise
             * it's a definition - we'll assume it's a prototype
             * though
             *---------------------------------------------------*/
            func->extType = TOKEN_FUNPRO;

            next = next->next;
            if (NULL == next)
               return;

            while ((TOKEN_COMMENT == next->extType) ||
                   (TOKEN_PREPROC == next->extType))
               {
               next = next->next;
```

```c

               if (NULL == next)
                  return;
               }

            if      (TOKEN_SCOLON == next->extType)
               func->extType = TOKEN_FUNPRO;
            else if (TOKEN_COMMA  == next->extType)
               func->extType = TOKEN_FUNPRO;
            else
               func->extType = TOKEN_FUNDEF;
            }
         }
      }
   }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * function that walks over hash table - for each thing in hash
 * table, see if it's a function - if not, release the strings
 * storage - otherwise add to the info hash table
 *------------------------------------------------------------------*/
static int CleanUpHashTableWalker(
   char **ident,
   Info  *info
   )
   {
   Function  func;
   Function *found;

   func.name = *ident;

   /*---------------------------------------------------------------
    * see if it's a function
    *---------------------------------------------------------------*/
   found = ListFind(info->funcTree,&func);

   /*---------------------------------------------------------------
    * if not found, release string storage
    *---------------------------------------------------------------*/
   if (!found)
      {
      free(*ident);
      return 0;
      }

   /*---------------------------------------------------------------
    * see if it's already in the info hash table
    *---------------------------------------------------------------*/
   if (HashFind(info->identHash,ident))
      {
      free(*ident);
      return 0;
      }

   /*---------------------------------------------------------------
    * if not, add it
    *---------------------------------------------------------------*/
   if (!HashAdd(info->identHash,ident))
      cPostError(1,"error adding identifier to global hash table");

   return 0;
   }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-------------------------------------------------------------------
 * parse the file for functions
 *-------------------------------------------------------------------*/
void cParse(
   File     *file,
   Info     *info
   )
 {
 void          *hTokenizer;
 Tok            *next;
 Tok            *prev;
 Tok            *fnc;
 static char   tmpIdent [MAX_IDENT_LEN + 1];
 char          *ptmpIdent  =   tmpIdent;
 char         **pptmpIdent = &ptmpIdent;
 char         **tempStr;

 /*-------------------------------------------------------------
  * read the file
  *-------------------------------------------------------------*/
 if (NULL == FileReadLines(info,file))
    cPostError(1,"unable to read file");

 file->identHash = HashCreate(sizeof(char *),
                              1000,
                              (HashFunc *)IdentHash,
                              (ListCompareFunc *)IdentCompare,
                              cPostNoMem);
 if (!file->identHash)
    cPostError(1,"error creating hash table");

 /*-----------------------------------------------------------
  * initialize tokenization
  *-----------------------------------------------------------*/
 hTokenizer       = CTokInit(GetBlockOfFile,file);
 if (!hTokenizer)
    cPostError(1,"error initializing tokenization");

 file->tokList    = NULL;
 next             = malloc(sizeof(Tok));
 if (!next)
    cPostError(1,"out of memory!!!");

 next->next       = NULL;
 next->nestParen = 0;
 next->nestBrace = 0;
 next->str        = NULL;

 prev = NULL;

 fnc = NULL;

 /*-----------------------------------------------------------
  * build list of tokens
  *-----------------------------------------------------------*/
 CTokGet(hTokenizer,&(next->tok));
 while (TOKEN_EOF != next->tok.type)
   {

    /*-----------------------------------------------------------
     * get a little more info
     *-----------------------------------------------------------*/
    next->extType = next->tok.type;

    if (TOKEN_OPER == next->extType)
       {
       switch(next->tok.ident[0])
          {
          case '{':
             next->extType = TOKEN_LBRACE;
             next->nestBrace++;

             if (next->nestParen)
                {
                fprintf(stderr,"unmatched ( on or before "
                               "line %ld\n",next->tok.line);
                next->nestParen = 0;
```

```
259                 }
260             break;
261
262         case '}':
263             next->extType = TOKEN_RBRACE;
264             next->nestBrace--;
265
266             if (next->nestParen)
267               {
268               fprintf(stderr,"unmatched ( on or before "
269                              "line %ld\n",next->tok.line);
270               next->nestParen = 0;
271               }
272             break;
273
274         case '(':
275             next->extType = TOKEN_LPAREN;
276             next->nestParen++;
277
278             /*-------------------------------------------------
279              * add the identifier before this, if there was one
280              *-------------------------------------------------*/
281             if (!fnc)
282                 break;
283
284             /*-------------------------------------------------------
285              * get string from hash or add it
286              *-------------------------------------------------------*/
287             tempStr = HashFind(file->identHash,pptmpIdent);
288             if (tempStr)
289                 fnc->str = *tempStr;
290             else
291               {
292               fnc->str = malloc(1+strlen(tmpIdent));
293               if (!fnc->str)
294                   cPostError(1,"out of memory!!!");
295
296               strcpy(fnc->str,tmpIdent);
297               if (!HashAdd(file->identHash,&(fnc->str)))
298                   cPostError(1,"error adding identifier to file hash table");
299               }
300
301             fnc = NULL;
302             break;
303
304         case ')':
305             next->extType = TOKEN_RPAREN;
306
307             if (next->nestParen)
308                 next->nestParen--;
309             else
310               {
311               fprintf(stderr,"unnested ) on "
312                              "line %ld\n",next->tok.line);
313               next->nestParen = 0;
314               }
315
316             break;
317
318         case ';':
319             next->extType = TOKEN_SCOLON;
320             break;
321
322         case ',':
323             next->extType = TOKEN_COMMA;
324             break;
325         }
326     }
327
328     else if (TOKEN_IDENT == next->extType)
329       {
330
331       if (IsKeyword(info,next->tok.ident))
332           next->extType = TOKEN_RESER;
333       else
334         {
335         strcpy(tmpIdent,next->tok.ident);
336         fnc = next;
337         }
338       }
339
```

```
340         /*-------------------------------------------------------------
341          * link into list
342          *-------------------------------------------------------------*/
343         if (NULL == file->tokList)
344            file->tokList = next;
345         else
346            prev->next = next;
347
348         /*-------------------------------------------------------------
349          * get next token
350          *-------------------------------------------------------------*/
351         prev            = next;
352         next            = malloc(sizeof(Tok));
353         if (!next)
354            cPostError(1,"out of memory!!!");
355
356         next->next      = NULL;
357         next->nestParen = prev->nestParen;
358         next->nestBrace = prev->nestBrace;
359         next->str       = NULL;
360
361         CTokGet(hTokenizer,&(next->tok));
362      }
363
364   /*----------------------------------------------------------------
365    * free the last hanging token
366    *----------------------------------------------------------------*/
367   free(next);
368
369   /*----------------------------------------------------------------
370    * terminate tokenization
371    *----------------------------------------------------------------*/
372   CTokTerm(hTokenizer);
373
374   /*----------------------------------------------------------------
375    * analyze the tokens
376    *----------------------------------------------------------------*/
377   AddFunctionInfoToTokens(file);
378   }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * clean up after parsing file for functions
 *------------------------------------------------------------------*/
void cParseDone(
   File     *file,
   Info     *info
   )
   {
   Tok *tok;
   Tok *next;
   int  i;

   /*------------------------------------------------------------------
    * walk over old hash table, adding functions to info hash table
    * and freeing non-function strings
    *------------------------------------------------------------------*/
   HashIterate(file->identHash,(ListIterateFunc *)CleanUpHashTableWalker,info);

   /*------------------------------------------------------------------
    * destroy the file hash table
    *------------------------------------------------------------------*/
   HashDestroy(file->identHash);

   /*------------------------------------------------------------------
    * now clean up token list
    *------------------------------------------------------------------*/
   tok = file->tokList;

   /*------------------------------------------------------------------
    * loop through tokens and free 'um
    *------------------------------------------------------------------*/
   while (tok)
      {
      next = tok->next;
      free(tok);
      tok = next;
      }

   /*------------------------------------------------------------------
    * release file storage
    *------------------------------------------------------------------*/
   for (i=0; i<(int)file->lines; i++)
      free(file->line[i]);

   free(file->line);
   }
```

```
/*-------------------------------------------------------------------
 * cpostutl.c : utilities for cPost
 *-------------------------------------------------------------------
 * 11-23-91 originally by Patrick J. Mueller
 * 12-03-92 converted from cBook to cPost
 *-------------------------------------------------------------------*/

#if defined(OPSYS_OS2) || defined(OPSYS_OS2V2)
    #define INCL_BASE
    #include <os2.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdarg.h>
#include <time.h>

#include "ctok.h"
#include "cpost.h"
#include "tokfile.h"
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------
 * compare two file names
 *-----------------------------------------------------------------*/
int FileNameCompare(
   File *file1,
   File *file2
   )
   {
   int   scmp;
   int   i;
   char *name1;
   char *name2;

   /*--------------------------------------------------------------
    * get the base file names
    *--------------------------------------------------------------*/
   name1 = malloc(1+strlen(file1->name));
   name2 = malloc(1+strlen(file2->name));

   if (!name1 || !name2)
      cPostError(1,"out of memory!!!");

   strcpy(name1,file1->name);
   strcpy(name2,file2->name);

   if (strchr(name1,'.')) *strchr(name1,'.') = 0;
   if (strchr(name2,'.')) *strchr(name2,'.') = 0;

   /*--------------------------------------------------------------
    * sort each category
    *--------------------------------------------------------------*/
   for (i=0; i<(int)strlen(info.oSort); i++)
      {
      /*-----------------------------------------------------------
       * sort by type
       *-----------------------------------------------------------*/
      if ('T' == info.oSort[i])
         {
         if (file1->type < file2->type)
            {
            free(name1);
            free(name2);
            return -1;
            }
         if (file1->type > file2->type)
            {
            free(name1);
            free(name2);
            return  1;
            }

         scmp = Stricmp(file1->ext,file2->ext);
         if (scmp)
            {
            free(name1);
            free(name2);
            return  scmp;
            }
         }

      /*-----------------------------------------------------------
       * sort by name
       *-----------------------------------------------------------*/
      else if ('N' == info.oSort[i])
         {
         /*--------------------------------------------------------
          * sort by name
          *--------------------------------------------------------*/
         scmp = Stricmp(file1->name,file2->name);
         if (scmp)
            {
            free(name1);
            free(name2);
            return  scmp;
```

```c
            }
        }
    }

    free(name1);
    free(name2);

    return 0;
}

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * compare two strings indirectly
 *------------------------------------------------------------------*/
int IdentCompare(
    char **str1,
    char **str2
    )
{
    return strcmp(*str1,*str2);
}

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * compare two function names
 *------------------------------------------------------------------*/
int FunctionNameCompare(
    Function *func1,
    Function *func2
    )
{
    int cmp;

    cmp = Stricmp(func1->name,func2->name);
    if (cmp) return cmp;

    cmp = strcmp(func1->name,func2->name);
    if (cmp) return cmp;

    return 0;
}

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * compare two function names (via indirection)
 *------------------------------------------------------------------*/
int FunctionNamePtrCompare(
    Function **func1,
    Function **func2
    )
{
    return FunctionNameCompare(*func1,*func2);
}
```

```
160
161 /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
162 /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
163
164 /*---------------------------------------------------------------------
165  * hash an identifier
166  *---------------------------------------------------------------------*/
167 int IdentHash(
168     char      **str,
169     int         hashSize
170     )
171     {
172     int    hash;
173     char *c;
174
175     for (hash=0, c=*str; *c; c++)
176         hash +=*c;
177
178     return hash % hashSize;
179     }
```

```

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-------------------------------------------------------------------
 * add a file to the list of files
 *-------------------------------------------------------------------*/
void FileAdd(
   Info      *info,
   List      *fileList,
   char      *name,
   char      *pathName,
   char      *fDate,
   char      *fTime
   )
   {
   File  file;
   char *type;
   char *test;
   char *ext;

   /*--------------------------------------------------------------
    * copy the name in
    *--------------------------------------------------------------*/
   file.name = malloc(1+strlen(name));
   if (NULL == file.name)
      cPostError(1,"out of memory!!!");

   strcpy(file.name,name);

   file.pathName = malloc(1+strlen(name)+strlen(pathName));
   if (NULL == file.pathName)
      cPostError(1,"out of memory!!!");

   strcpy(file.pathName,pathName);
   strcat(file.pathName,name);

   strcpy(file.date,fDate);
   strcpy(file.time,fTime);

   /*--------------------------------------------------------------
    * get the extension
    *--------------------------------------------------------------*/
   ext = strchr(file.name,'.');
   if (!ext)
      file.ext = "";
   else
      {
      ext++;
      file.ext = malloc(1+strlen(ext));
      if (!file.ext)
         cPostError(1,"out of memory!!!");

      strcpy(file.ext,ext);

      /*-----------------------------------------------------------
       * remove . and anything following it
       *-----------------------------------------------------------*/
      if (strchr(file.ext,'.'))
         *strchr(file.ext,'.') = '\0';
      }

   /*--------------------------------------------------------------
    * default the type to 'other'
    *--------------------------------------------------------------*/
   file.type = 2;

   /*--------------------------------------------------------------
    * see if it's an 'H' file
    *--------------------------------------------------------------*/
   type = malloc(1+strlen(info->oHtype));
   if (!type)
      cPostError(1,"out of memory!!!");

   strcpy(type,info->oHtype);
   test = strtok(type,", ");
   while (test)
```

```
257         {
258         if (!Stricmp(file.ext,test))
259            file.type = 0;
260
261         test = strtok(NULL,", ");
262         }
263
264      free(type);
265
266      /*-----------------------------------------------------------------
267       * see if it's a 'C' file
268       *-----------------------------------------------------------------*/
269      type = malloc(1+strlen(info->oCtype));
270      if (!type)
271         cPostError(1,"out of memory!!!");
272
273      strcpy(type,info->oCtype);
274      test = strtok(type,", ");
275      while (test)
276         {
277         if (!Stricmp(file.ext,test))
278            file.type = 1;
279
280         test = strtok(NULL,", ");
281         }
282
283      free(type);
284
285      /*-----------------------------------------------------------------
286       * set rest of stuff
287       *-----------------------------------------------------------------*/
288      file.line      = NULL;
289      file.lines     = 0L;
290      file.cline     = 0L;
291      file.tempName  = NULL;
292      file.breakList = NULL;
293
294      file.funcDefList = ListCreate(sizeof(Function *),
295                                    (ListCompareFunc *)FunctionNamePtrCompare,
296                                    cPostNoMem);
297      if (!file.funcDefList)
298         cPostError(1,"error creating function definition list");
299
300      file.funcProList = ListCreate(sizeof(Function *),
301                                    (ListCompareFunc *)FunctionNamePtrCompare,
302                                    cPostNoMem);
303      if (!file.funcProList)
304         cPostError(1,"error creating function prototype list");
305
306      /*-----------------------------------------------------------------
307       * now add it to the list
308       *-----------------------------------------------------------------*/
309      if (ListFind(fileList,&file))
310         return;
311
312      if (ListAdd(fileList,&file))
313         return;
314
315      /*-----------------------------------------------------------------
316       * otherwise, error adding it
317       *-----------------------------------------------------------------*/
318      cPostError(1,"error adding file %s to file list",name);
319      }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

#if defined(OPSYS_OS2) || defined(OPSYS_OS2V2)
/*----------------------------------------------------------------------
 * OS/2 version - add all the files in a file spec to the list
 *----------------------------------------------------------------------*/
void FileSpecAddOS2(
   Info      *info,
   List      *fileList,
   char      *fileSpec
   )
   {
   USHORT       rc;
   HDIR         hDir;
#if defined(OPSYS_OS2V2)
   FILEFINDBUF3 ffbuf;
   ULONG        attr;
   ULONG        cnt;
#else
   FILEFINDBUF  ffbuf;
   USHORT       attr;
   USHORT       cnt;
#endif
   int              files;
   static UCHAR    pathName[261];
   char             szTime[9];
   char             szDate[9];

   /*-------------------------------------------------------------------
    * get path name of spec
    *-------------------------------------------------------------------*/
#if defined(OPSYS_OS2V2)
   rc = DosQueryPathInfo(fileSpec,5,pathName,sizeof(pathName));
#else
   rc = DosQPathInfo(fileSpec,5,pathName,sizeof(pathName),0);
#endif
   if (rc)
      cPostError(1,"error getting path for %s",fileSpec);

   /*-------------------------------------------------------------------
    * convert slashes to back slashes
    *-------------------------------------------------------------------*/
   while (strchr(pathName,'/'))
      *strchr(pathName,'/') = '\\';

   *(strrchr(pathName,'\\') + 1) = '\0';

   /*-------------------------------------------------------------------
    * get the first file
    *-------------------------------------------------------------------*/
   hDir = 0xFFFF;
   attr = 0;
   cnt  = 1;

#if defined(OPSYS_OS2V2)
   rc = DosFindFirst(fileSpec,&hDir,attr,&ffbuf,sizeof(ffbuf), &cnt,1);
#else
   rc = DosFindFirst(fileSpec,&hDir,attr,&ffbuf,sizeof(ffbuf), &cnt,0);
#endif

   /*-------------------------------------------------------------------
    * continue while we keep getting files
    *-------------------------------------------------------------------*/
   for (files=0; 0 == rc; files++)
      {
      sprintf(szDate,"%2.2d/%2.2d/%2.2d",
              (int) ffbuf.fdateLastWrite.month,
              (int) ffbuf.fdateLastWrite.day,
              (int) ffbuf.fdateLastWrite.year+80);

      sprintf(szTime,"%2.2d:%2.2d:%2.2d",
              (int) ffbuf.ftimeLastWrite.hours,
              (int) ffbuf.ftimeLastWrite.minutes,
              (int) ffbuf.ftimeLastWrite.twosecs * 2);

```

```
397         if (!IsTempFileName(ffbuf.achName))
398            FileAdd(info,fileList,ffbuf.achName,pathName,szDate,szTime);
399
400         /*----------------------------------------------------------
401          * get next file
402          *----------------------------------------------------------*/
403         rc = DosFindNext(hDir,&ffbuf,sizeof(ffbuf),&cnt);
404         }
405
406      rc = DosFindClose(hDir);
407      }
```

```
408
409 #elif defined(OPSYS_CMS)
410 /*-------------------------------------------------------------------
411  * cms version - add all the files in a file spec to the list
412  *-------------------------------------------------------------------*/
413 void FileSpecAddCMS(
414    Info       *info,
415    List       *fileList,
416    char       *fileSpec
417    )
418    {
419    int     rc;
420    FILE   *stack;
421    int     num;
422    char   *copy;
423    char   *buffer;
424    char    fileName[21];
425    char   *fileDate;
426    char   *fileTime;
427    int     i;
428
429 #define BUFFER_LEN 1000
430
431    /*-------------------------------------------------------------------
432     * make copy of spec, and translate '.' to ' '
433     *-------------------------------------------------------------------*/
434    copy = malloc(1+strlen(fileSpec));
435    if (!copy)
436       cPostError(1,"out of memory!!!");
437
438    strcpy(copy,fileSpec);
439    fileSpec = copy;
440
441    /*-------------------------------------------------------------------
442     * translate '.' to ' '
443     *-------------------------------------------------------------------*/
444    while (strchr(fileSpec,'.'))
445       *strchr(fileSpec,'.') = ' ';
446
447    /*-------------------------------------------------------------------
448     * build command string
449     *-------------------------------------------------------------------*/
450    buffer = malloc(BUFFER_LEN);
451    if (!buffer)
452       cPostError(1,"out of memory!!!");
453
454    strcpy(buffer,"LISTFILE ");
455    strcat(buffer,fileSpec);
456    strcat(buffer," ( NOH STACK DATE");
457
458    /*-------------------------------------------------------------------
459     * set high water mark
460     *-------------------------------------------------------------------*/
461    system("MAKEBUF");
462
463    /*-------------------------------------------------------------------
464     * run command
465     *-------------------------------------------------------------------*/
466    rc = system(buffer);
467    if (rc)
468       {
469       cPostError(rc,"return code %d from '%s'",rc,buffer);
470       exit(rc);
471       }
472
473    /*-------------------------------------------------------------------
474     * see how many stacked
475     *-------------------------------------------------------------------*/
476    num = system("SENTRIES");
477
478    /*-------------------------------------------------------------------
479     * open the stack
480     *-------------------------------------------------------------------*/
481    stack = fopen("*","r");
482    if (!stack)
483       {
484       cPostError(1,"error opening stack for reading");
```

```
485         exit(1);
486         }
487
488     /*-----------------------------------------------------------
489      * read the stack, add files
490      *-----------------------------------------------------------*/
491     while (num--)
492         {
493         fgets(buffer,BUFFER_LEN,stack);
494         if ('\n' == buffer[strlen(buffer)-1])
495             buffer[strlen(buffer)-1] = '\0';
496
497         /*-------------------------------------------------------
498          * get the file name
499          *-------------------------------------------------------*/
500         copy = strtok(buffer," ");
501         strcpy(fileName,copy);
502         strcat(fileName,".");
503
504         copy = strtok(NULL," ");
505         strcat(fileName,copy);
506         strcat(fileName,".");
507
508         copy = strtok(NULL," ");
509         strcat(fileName,copy);
510
511         /*-------------------------------------------------------
512          * get the date and time
513          *-------------------------------------------------------*/
514         for (i=0; i<4; i++)
515             strtok(NULL," ");
516
517         fileDate = strtok(NULL," ");
518         fileTime = strtok(NULL," ");
519
520         /*-------------------------------------------------------
521          * add file
522          *-------------------------------------------------------*/
523         FileAdd(info,fileList,fileName,"",fileDate,fileTime);
524         }
525
526     /*-----------------------------------------------------------
527      * close stack, free memory, leave
528      *-----------------------------------------------------------*/
529     fclose(stack);
530     free(fileSpec);
531     free(buffer);
532
533     }
```

```
534
535 /*---------------------------------------------------------------------
536  * other operating systems (AIX, DOS, ???)
537  *---------------------------------------------------------------------*/
538 #else
539
540 #include <sys/types.h>
541 #include <sys/stat.h>
542
543 /*---------------------------------------------------------------------
544  * get file date and time
545  *---------------------------------------------------------------------*/
546 void getFileDateTime(
547    char *fileName,
548    char *fileDate,
549    char *fileTime
550    )
551 {
552 struct stat  s;
553 struct tm   *t;
554
555 *fileDate = 0;
556 *fileTime = 0;
557
558 if (stat(fileName,&s))
559    return;
560
561 t = localtime(&(s.st_mtime));
562
563 sprintf(fileDate,"%2.2d/%2.2d/%2.2d",
564         (int) t->tm_mon + 1,
565         (int) t->tm_mday,
566         (int) t->tm_year % 100);
567
568 sprintf(fileTime,"%2.2d:%2.2d:%2.2d",
569         (int) t->tm_hour,
570         (int) t->tm_min,
571         (int) t->tm_sec);
572 }
```

```
573
574  /*-------------------------------------------------------------------
575   * generic version - add all the files in a file spec to the list
576   *   this will work for AIX
577   *-------------------------------------------------------------------*/
578  void FileSpecAddGeneric(
579     Info       *info,
580     List       *fileList,
581     char       *fileSpec
582     )
583     {
584     char         *pathName;
585     char         *fileName;
586     struct stat  statBuff;
587     char          dateBuff[9];
588     char          timeBuff[9];
589
590     /*-----------------------------------------------------------
591      * get area for path name
592      *-----------------------------------------------------------*/
593     if (!strchr(fileSpec,'/'))
594        {
595        pathName = "";
596        fileName = fileSpec;
597        }
598
599     else
600        {
601        /*-----------------------------------------------------------
602         * convert back slashes to slashes
603         *-----------------------------------------------------------*/
604        while (strchr(fileSpec,'\\'))
605           *strchr(fileSpec,'\\') = '/';
606
607        /*-----------------------------------------------------------
608         * get path part of file spec
609         *-----------------------------------------------------------*/
610        pathName = malloc(1+strlen(fileSpec));
611        if (!pathName)
612           cPostError(1,"out of memory!!!");
613
614        strcpy(pathName,fileSpec);
615
616        *(strrchr(pathName,'/') + 1) = '\0';
617
618        /*-----------------------------------------------------------
619         * get filename part of file spec
620         *-----------------------------------------------------------*/
621        fileName = malloc(1+strlen(fileSpec));
622        if (!fileName)
623           cPostError(1,"out of memory!!!");
624
625        strcpy(fileName,fileSpec);
626        fileName = strrchr(fileName,'/') + 1;
627        }
628
629     /*-----------------------------------------------------------
630      * get file date and time
631      *-----------------------------------------------------------*/
632     getFileDateTime(fileSpec,dateBuff,timeBuff);
633
634     /*-----------------------------------------------------------
635      * add file
636      *-----------------------------------------------------------*/
637     FileAdd(info,fileList,fileName,pathName,dateBuff,timeBuff);
638     }
```

```
#endif

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*---------------------------------------------------------------------
 * the router for FileSpecAdd
 *---------------------------------------------------------------------*/
void FileSpecAdd(
   Info      *info,
   List      *fileList,
   char      *fileSpec
   )
   {
   TokFileInfo tfi;


   /*------------------------------------------------------------------
    * check for @ files
    *------------------------------------------------------------------*/
   if (('@' == fileSpec[0]) && (1 != strlen(fileSpec)))
      {
      fileSpec++;
      tfi = TokFileOpen(fileSpec);
      if (!tfi)
         {
         cPostError(0,"error opening file '%s' for reading",fileSpec);
         return;
         }

      /*---------------------------------------------------------------
       * add filenames from file
       *---------------------------------------------------------------*/
      while (NULL != (fileSpec = TokFileNext(tfi)))
         FileSpecAdd(info,fileList,fileSpec);

      return;
      }

   /*------------------------------------------------------------------
    * call the op/sys dependant function
    *------------------------------------------------------------------*/

#if defined(OPSYS_OS2) || defined(OPSYS_OS2V2)

   FileSpecAddOS2(info,fileList,fileSpec);

#elif defined(OPSYS_CMS)

   FileSpecAddCMS(info,fileList,fileSpec);

#else

   FileSpecAddGeneric(info,fileList,fileSpec);

#endif
   }
```

```
697
698 /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
699 /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
700
701 /*----------------------------------------------------------------------
702  * count occurances of a character in a string
703  *----------------------------------------------------------------------*/
704 static int strcnt(
705    char  c,
706    char *string
707    )
708    {
709    int   cnt;
710
711    cnt    = 0;
712    string = strchr(string,c);
713
714    while (string)
715       {
716       cnt++;
717       string++;
718       string = strchr(string,c);
719       }
720
721    return cnt;
722    }
```

```

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * expand tabs
 *------------------------------------------------------------------*/
static char *ExpandTabs(
   char *line,
   int   expand
   )
   {
   int   tabs;
   char *newLine;
   int   lineLen;
   int   col;
   char *c1;
   char *c2;

   /*------------------------------------------------------------------
    * get # of tabs - if no tabs, return line intact
    *------------------------------------------------------------------*/
   tabs = strcnt('\t',line);
   if (!tabs)
      return line;

   /*------------------------------------------------------------------
    * otherwise, allocate space for new line
    *------------------------------------------------------------------*/
   lineLen = strlen(line);
   newLine = malloc(lineLen + 1 + expand * tabs);
   if (!newLine)
      cPostError(1,"out of memory!!!");

   memset(newLine,0,lineLen + 1 + expand * tabs);

   /*------------------------------------------------------------------
    * copy old string to new string, expanding tabs as you go
    *------------------------------------------------------------------*/
   col = 1;
   c1  = line;
   c2  = newLine;
   while (*c1)
      {
      /*------------------------------------------------------------------
       * copy non-tab chars into new string
       *------------------------------------------------------------------*/
      if ('\t' != *c1)
         {
         *c2++ = *c1++;
         col++;
         }

      else
         {
         c1++;
/*------------------------------------------------------------------
 * art roberts identified this bug in the code and also supplied a
 * fix (below).
 *------------------------------------------------------------------*/
#if 0
         for (i = col%expand; i < (expand+1); i++)
            {
            *c2++ = ' ';
            col++;
            }
#else
         do
            {
            *c2++ = ' ';
            }
         while ((col++ % expand) != 0);
#endif
         }
      }

   /*------------------------------------------------------------------
    * free original line, return new line
    *------------------------------------------------------------------*/
   free(line);
   return newLine;
```

```
804     }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * read file into array of lines
 *------------------------------------------------------------------*/
File *FileReadLines(
   Info *info,
   File *file
   )
   {
   static char    buffer [MAX_LINE_LEN];
   FILE             *hFile;
   unsigned long   lines;
   char             *line;

   /*------------------------------------------------------------
    * initialize
    *------------------------------------------------------------*/
   file->line        = NULL;
   file->lines       = 0;
   file->cline       = 0L;
   file->maxLineLen = 0;
   memset(buffer,'\0',sizeof(buffer));

   /*------------------------------------------------------------
    * open file
    *------------------------------------------------------------*/
   if (file->tempName)
      {
      hFile = fopen(file->tempName,"r");
      if (NULL == hFile)
         cPostError(1,"error opening file %s for reading",file->tempName);
      }

   else
      {
      hFile = fopen(file->pathName,"r");
      if (NULL == hFile)
         cPostError(1,"error opening file %s for reading",file->pathName);
      }

   /*------------------------------------------------------------
    * buffer input
    *------------------------------------------------------------*/
/* setvbuf(hFile,NULL,_IOFBF,32000); */

   /*------------------------------------------------------------
    * allocate area for lines
    *------------------------------------------------------------*/
   lines = FILE_LINES;
   file->line = malloc(((int)lines)*sizeof(char *));
   if (NULL == file->line)
      cPostError(1,"out of memory!!!");

   /*------------------------------------------------------------
    * loop for each line in the file
    *------------------------------------------------------------*/
   for (file->lines=0;;file->lines++)
      {
      /*---------------------------------------------------------
       * reallocate buffer if we need to
       *---------------------------------------------------------*/
      if (file->lines >= lines)
         {
         lines += FILE_LINES;
         file->line    = realloc(file->line,((int)lines)*sizeof(char *));
         if (NULL == file->line)
            cPostError(1,"out of memory!!!");
         }

      /*---------------------------------------------------------
       * read a line
       *---------------------------------------------------------*/
#if defined(DO_IT_NO_MORE)
      if (info->indent1)
         {
         buffer[0] = ' ';
         if (!fgets(&(buffer[1]),MAX_LINE_LEN-1,hFile))
```

```
                break;
            }

        else
#endif
            {
            if (!fgets(buffer,MAX_LINE_LEN,hFile))
                break;
            }


        /*----------------------------------------------------------
         * malloc space for new line
         *----------------------------------------------------------*/
        line = malloc(1+strlen(buffer));
        if (!line)
            cPostError(1,"out of memory!!!");

        /*----------------------------------------------------------
         * copy line we just got into new line
         *----------------------------------------------------------*/
        strcpy(line,buffer);

        /*-------------------------------------------------------------
         * expand tabs
         *-------------------------------------------------------------*/
        line = ExpandTabs(line,info->oTabs);

        /*----------------------------------------------------------
         * set vars
         *----------------------------------------------------------*/
        file->line[file->lines] = line;
        file->maxLineLen = max(file->maxLineLen,(int)strlen(line));
        }

    /*-------------------------------------------------------------
     * close the file
     *-------------------------------------------------------------*/
    fclose(hFile);

    return file;
    }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-------------------------------------------------------------------
 * return bytes from the file, conveniently a line at a time
 *-------------------------------------------------------------------*/
unsigned long GetBlockOfFile(
    void   *readInfo,
    char  **buffer
    )
    {
    File *file;
    file = readInfo;

    if (file->cline >= file->lines)
        return 0L;

    *buffer = file->line[file->cline++];
    return (unsigned long) strlen(*buffer);
    }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------------
 * get function (add if needed) from global function table
 *-----------------------------------------------------------------------*/
Function *GetFunction(
   Info          *info,
   char          *name
   )
   {
   Function  func;
   Function *found;

   func.name = name;

   /*------------------------------------------------------------------
    * look for function
    *------------------------------------------------------------------*/
   found = ListFind(info->funcTree,&func);
   if (found)
      return found;

   /*------------------------------------------------------------------
    * fill in fields if not found
    *------------------------------------------------------------------*/
   func.id           = 0;
   func.spotted      = 0;
   func.callsList    = ListCreate(sizeof(Function *),
                               (ListCompareFunc *)FunctionNamePtrCompare,
                               cPostNoMem);
   if (!func.callsList)
      cPostError(1,"error creating function calls list");

   func.calledByList = ListCreate(sizeof(Function *),
                               (ListCompareFunc *)FunctionNamePtrCompare,
                               cPostNoMem);
   if (!func.calledByList)
      cPostError(1,"error creating function called by list");

   func.name         = malloc(1+strlen(name));
   if (!func.name)
      cPostError(1,"out of memory!!!");

   strcpy(func.name,name);

   func.lineNo       = 0;
   func.fileName     = "";

   /*------------------------------------------------------------------
    * add it, return pointer
    *------------------------------------------------------------------*/
   if (!ListAdd(info->funcTree,&func))
      cPostError(1,"error adding function to function list");

   found = ListFind(info->funcTree,&func);
   if (!found)
      cPostError(1,"error retrieving function from function list");

   return found;
   }
```

```c
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------
 * prinf function table entry
 *-----------------------------------------------------------------*/
static int PrintFunctionTableEntry(
   Function **func,
   Info      *info
   )
   {
   fprintf(info->oFile,":hp2.%s:ehp2.",(*func)->name);

   if ('\0' != *((*func)->fileName))
      {
      fprintf(info->oFile," (%s, page :spotref refid=sp%4.4d.)\n",
                     (*func)->fileName,(*func)->id);
      }

   else
      fprintf(info->oFile,"\n");

   info->count1++;
   return 0;
   }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------
 * print function pointer information
 *-----------------------------------------------------------------*/
int PrintFunctionPtrInfo(
   Function **func,
   Info      *info
   )
   {
   return PrintFunctionInfo(*func,info);
   }
```

```
1050
1051  /*-------------------------------------------------------------------
1052   * print function information
1053   *-------------------------------------------------------------------*/
1054  int PrintFunctionInfo(
1055     Function  *func,
1056     Info      *info
1057     )
1058    {
1059
1060     /*-------------------------------------------------------------------
1061      * print table definition
1062      *-------------------------------------------------------------------*/
1063     fprintf(info->oFile,":table refid=reftbl.\n");
1064
1065     /*-------------------------------------------------------------------
1066      * print function name
1067      *-------------------------------------------------------------------*/
1068     fprintf(info->oFile,":row.:c 5.&fn%4.4d.",func->id);
1069
1070     if ('\0' != *(func->fileName))
1071       {
1072       fprintf(info->oFile," (%s, page :spotref refid=sp%4.4d.)\n",
1073                     func->fileName,func->id);
1074       }
1075
1076     else
1077        fprintf(info->oFile,"\n");
1078
1079     /*-------------------------------------------------------------------
1080      * print calls list
1081      *-------------------------------------------------------------------*/
1082     if (ListCount(func->callsList))
1083       {
1084       fprintf(info->oFile,":c 1.calls\n:c 2.");
1085
1086       info->count1 = 1;
1087       info->count2 = ListCount(func->callsList);
1088
1089       ListIterate(func->callsList,
1090                   (ListIterateFunc *)PrintFunctionTableEntry,
1091                   info);
1092       }
1093
1094     /*-------------------------------------------------------------------
1095      * print called by list
1096      *-------------------------------------------------------------------*/
1097     if (ListCount(func->calledByList))
1098       {
1099       fprintf(info->oFile,":c 3.called\nby\n:c 4.");
1100
1101       info->count1 = 1;
1102       info->count2 = ListCount(func->calledByList);
1103
1104       ListIterate(func->calledByList,
1105                   (ListIterateFunc *)PrintFunctionTableEntry,
1106                   info);
1107       }
1108
1109     fprintf(info->oFile,":etable.\n");
1110
1111     return 0;
1112    }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * generate a temporary file name
 *------------------------------------------------------------------*/
char *TempFileName(
   FILE **file,
   Info  *info
   )
   {
   char        *name;
   static int  counter = 0;

   name = malloc(13 + 1 + strlen(info->oTemp));
   if (!name)
      cPostError(1,"out of memory!!!");

   for(;;)
      {

#if defined(OPSYS_CMS)
      if (*(info->oTemp))
         sprintf(name,"cps%.5d.tmp.%c",counter++,*(info->oTemp));
      else
         sprintf(name,"cps%.5d.tmp",counter++);
#else
      sprintf(name,"%scps%.5d.tmp",info->oTemp,counter++);
#endif

      /*------------------------------------------------------------------
       * open temp file
       *------------------------------------------------------------------*/
      *file = fopen(name,"r");
      if (NULL == *file)
         {
         *file = fopen(name,"w");
         if (NULL != *file)
            return name;
         else
            {
            cPostError(0,"error opening temp file '%s' for writing",name);
            cPostError(1,"check -y value for writable temporary path");
            }
         }

      fclose(*file);
      }

   /*------------------------------------------------------------------
    * can't really get here anymore
    *------------------------------------------------------------------*/
   cPostError(1,"can't create temp file");
   return NULL;
   }

/*------------------------------------------------------------------
 * see if a file is one of our temporary files
 *------------------------------------------------------------------*/
int IsTempFileName(
   char *name
   )
   {
   if (Memicmp(name,"cps",3))
      return 0;

   if (Memicmp(name+8,".tmp",4))
      return 0;

   return 1;
   }
```

```
1185
1186  /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
1187  /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
1188
1189  /*---------------------------------------------------------------------
1190   * compare strings case insensitively
1191   *---------------------------------------------------------------------*/
1192  int Stricmp(
1193     char *str1,
1194     char *str2
1195     )
1196    {
1197    char c1;
1198    char c2;
1199
1200    while (*str1 && *str2)
1201      {
1202      c1 = (char) toupper(*str1);
1203      c2 = (char) toupper(*str2);
1204
1205      if (c1 < c2)
1206         return -1;
1207      else if (c1 > c2)
1208         return  1;
1209
1210      str1++; str2++;
1211      }
1212
1213    if (!*str1 && !*str2)
1214       return 0;
1215
1216    if (*str1)
1217       return  1;
1218    else
1219       return -1;
1220    }
1221
1222  /*---------------------------------------------------------------------
1223   * compare strings case insensitively
1224   *---------------------------------------------------------------------*/
1225  int Memicmp(
1226     char *str1,
1227     char *str2,
1228     int   len
1229     )
1230    {
1231    char c1;
1232    char c2;
1233    int  i;
1234
1235    for (i=0; i<len; i++)
1236      {
1237      c1 = (char) toupper(*str1);
1238      c2 = (char) toupper(*str2);
1239
1240      if (c1 < c2)
1241         return -1;
1242      else if (c1 > c2)
1243         return  1;
1244
1245      str1++; str2++;
1246      }
1247
1248    return 0;
1249    }
```

```
1250
1251  /*---------------------------------------------------------------------
1252   * copy a string
1253   *---------------------------------------------------------------------*/
1254  char *Strcopy(
1255     char *str
1256     )
1257     {
1258     char *result;
1259     char *next;
1260
1261     next = result = malloc(1+strlen(str));
1262
1263     while (*str)
1264        {
1265        *next = *str;
1266        str++;
1267        next++;
1268        }
1269
1270     return result;
1271     }
1272
1273  /*---------------------------------------------------------------------
1274   * upper case a string
1275   *---------------------------------------------------------------------*/
1276  char *Strupr(
1277     char *str
1278     )
1279     {
1280     char *result;
1281     char *next;
1282
1283     next = result = Strcopy(str);
1284
1285     while (*next)
1286        {
1287        *next = (char) toupper(*next);
1288        next++;
1289        }
1290
1291     return result;
1292     }
1293
1294  #if 0
1295  /*---------------------------------------------------------------------
1296   * reverse a string
1297   *---------------------------------------------------------------------*/
1298  char *Strrev(
1299     char *str
1300     )
1301     {
1302     int    len;
1303     char   temp;
1304     int    i;
1305
1306     len = strlen(str);
1307
1308     for (i=0; i<(len+1)/2-1; i++)
1309        {
1310        temp          = str[i];
1311        str[i]        = str[len-1-i];
1312        str[len-i-1] = temp;
1313        }
1314
1315     return str;
1316     }
```

```c
#endif

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * called when out of memory
 *------------------------------------------------------------------*/
void cPostNoMem(void)
  {
  cPostError(1,"out of memory!!!");
  }
```

```
/*-------------------------------------------------------------------
 * ctok : C language tokenizer
 *-------------------------------------------------------------------
 * 10-01-91 Patrick J. Mueller
 *-------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "ctok.h"

/*-------------------------------------------------------------------
 * is a character a valid character in a C identifier
 *-------------------------------------------------------------------*/
#define isCsymbol(c) (isalnum(c) || ('_' == c))

/*-------------------------------------------------------------------
 * typedefs
 *-------------------------------------------------------------------*/
typedef struct
    {
    int             eof;
    char           *buffer;
    long            bufferLen;
    long            bufferInd;
    long            fileOffs;
    long            line;
    int             unGetChar;
    int             unGetReady;
    long            tokOffs;
    long            tokLen;
    CTokRead        readFunc;
    void           *readInfo;
    char            ident[MAX_IDENT_LEN+1];
    } CTokInfo;
```

```
38
39  /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
40  /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
41
42  /*---------------------------------------------------------------
43   * get next char from file
44   *---------------------------------------------------------------*/
45  void GetNextChar(
46     int          *c,
47     CTokInfo     *cti
48     )
49    {
50    cti->fileOffs++;
51
52    /*---------------------------------------------------------------
53     * check for end of file
54     *---------------------------------------------------------------*/
55    if (cti->eof)
56       {
57       *c = EOF;
58       return;
59       }
60
61    /*---------------------------------------------------------------
62     * check for a char in the unget holder
63     *---------------------------------------------------------------*/
64    if (cti->unGetReady)
65       {
66       cti->unGetReady = 0;
67       *c = cti->unGetChar;
68
69       if ('\n' == *c)
70          cti->line++;
71       return;
72       }
73
74    /*---------------------------------------------------------------
75     * see if we need to read another buffer
76     *---------------------------------------------------------------*/
77    if (cti->bufferInd == cti->bufferLen)
78       {
79       cti->bufferLen = cti->readFunc(cti->readInfo,&(cti->buffer));
80       cti->bufferInd = 0L;
81
82       if (0L == cti->bufferLen)
83          {
84          *c = EOF;
85          cti->eof = 1;
86          return;
87          }
88       }
89
90    /*---------------------------------------------------------------
91     * read character from buffer
92     *---------------------------------------------------------------*/
93    *c = cti->buffer[cti->bufferInd++];
94
95    if ('\n' == *c)
96       cti->line++;
97
98    return;
99    }
```

```c

/*-------------------------------------------------------------------
 * put back last char from file
 *-------------------------------------------------------------------*/
void UnGetNextChar(
   int           c,
   CTokInfo     *cti
   )
   {
   cti->fileOffs--;

   cti->unGetChar  = c;
   cti->unGetReady = 1;

   if ('\n' == c)
      cti->line--;
   }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-------------------------------------------------------------------
 * read a C character constant or string
 *-------------------------------------------------------------------*/
static void ReadString(
   CTokInfo     *cti,
   int           c
   )
   {
   int stop;

   /*----------------------------------------------------------------
    * the character passed in is ' or ", and it is the character that
    * signifies the end of the string
    *----------------------------------------------------------------*/
   stop = c;

   /*----------------------------------------------------------------
    * keep going until we hit our stop character
    *----------------------------------------------------------------*/
   GetNextChar(&c,cti);
   while (stop != c)
      {
      /*-------------------------------------------------------------
       * for a \, inhale next character
       *-------------------------------------------------------------*/
      if ('\\' == c)
         GetNextChar(&c,cti);

      /*-------------------------------------------------------------
       * for EOF, break
       *-------------------------------------------------------------*/
      if (EOF == c)
         break;

      GetNextChar(&c,cti);
      }

   return;
   }
```

```
160
161 /*-------------------------------------------------------------------
162  * read a C comment
163  *-------------------------------------------------------------------*/
164 static void ReadComment(
165    CTokInfo    *cti
166    )
167 {
168 int c;
169
170  /*-------------------------------------------------------------------
171   * loop until end of file (or return in middle)
172   *-------------------------------------------------------------------*/
173  GetNextChar(&c,cti);
174  while (EOF != c)
175  {
176
177  /*-------------------------------------------------------------------
178   * if not *, just get next character
179   *-------------------------------------------------------------------*/
180  if ('*' != c)
181     GetNextChar(&c,cti);
182
183  /*-------------------------------------------------------------------
184   * got a * - see if next is /
185   *-------------------------------------------------------------------*/
186  else
187     {
188  /*----------------------------------------------------------
189   * if next is /, return
190   *----------------------------------------------------------*/
191  GetNextChar(&c,cti);
192  if ('/'  == c)
193     return;
194     }
195
196  }
197
198 return;
199 }
200
201 /*-------------------------------------------------------------------
202  * read a C++ style comment
203  *-------------------------------------------------------------------*/
204 static void ReadCppComment(
205    CTokInfo    *cti
206    )
207 {
208 int c;
209
210  /*-------------------------------------------------------------------
211   * loop until end of line or end of file
212   *-------------------------------------------------------------------*/
213  GetNextChar(&c,cti);
214
215  while ((EOF != c) && ('\n' != c))
216     GetNextChar(&c,cti);
217
218  UnGetNextChar(c,cti);
219  return;
220 }
```

```
221
222 /*-------------------------------------------------------------------
223  * read an identifier
224  *-------------------------------------------------------------------*/
225 static void ReadIdent(
226    CTokInfo    *cti,
227    int         c
228    )
229    {
230    int identLen;
231
232    /*-------------------------------------------------------------------
233     * initialize length and stick first char in
234     *-------------------------------------------------------------------*/
235    identLen = 0;
236    cti->ident[identLen++] = (char) c;
237
238    /*-------------------------------------------------------------------
239     * while still a valid symbol character ...
240     *-------------------------------------------------------------------*/
241    GetNextChar(&c,cti);
242    while (isCsymbol(c))
243       {
244       /*-------------------------------------------------------------------
245        * make sure we got enough room, then stick it in
246        *-------------------------------------------------------------------*/
247       if (identLen < MAX_IDENT_LEN)
248          cti->ident[identLen++] = (char) c;
249
250       GetNextChar(&c,cti);
251       }
252
253    /*-------------------------------------------------------------------
254     * finish up identifier, put last character back
255     *-------------------------------------------------------------------*/
256    cti->ident[identLen] = '\0';
257    UnGetNextChar(c,cti);
258    }
259
260 /*-------------------------------------------------------------------
261  * read a number
262  *-------------------------------------------------------------------*/
263 static void ReadNumber(
264    CTokInfo    *cti,
265    int         c
266    )
267    {
268
269    /*-------------------------------------------------------------------
270     * while still a valid number character ...
271     *-------------------------------------------------------------------*/
272    GetNextChar(&c,cti);
273    while (isalnum(c))
274       GetNextChar(&c,cti);
275
276    /*-------------------------------------------------------------------
277     * put last character back
278     *-------------------------------------------------------------------*/
279    UnGetNextChar(c,cti);
280    }
```

```
281
282 /*-------------------------------------------------------------
283  * read a preprocessor statement
284  *-------------------------------------------------------------*/
285 static void ReadPreprocessor(
286    CTokInfo    *cti
287    )
288 {
289 int c;
290
291 /*-------------------------------------------------------------
292  * loop until end of file (or return in middle)
293  *-------------------------------------------------------------*/
294 GetNextChar(&c,cti);
295 while (EOF != c)
296    {
297    /*----------------------------------------------------------
298     * if we found a newline, leave
299     *----------------------------------------------------------*/
300    if ('\n' == c)
301       {
302       UnGetNextChar(c,cti);
303       return;
304       }
305
306    /*----------------------------------------------------------
307     * if we got anything but a \, eat it
308     *----------------------------------------------------------*/
309    else if ('\\' != c)
310       GetNextChar(&c,cti);
311
312    /*----------------------------------------------------------
313     * got a \ - see if next is \n
314     *----------------------------------------------------------*/
315    else
316       {
317       /*-------------------------------------------------------
318        * if next isn't \n, start at top of loop
319        *-------------------------------------------------------*/
320       GetNextChar(&c,cti);
321
322       /*-------------------------------------------------------
323        * skip over white space first
324        *-------------------------------------------------------*/
325       while (isspace(c) && ('\n' != c))
326          GetNextChar(&c,cti);
327
328       if ('\n' != c)
329          continue;
330
331       /*-------------------------------------------------------
332        * if it is a \n, read next char and continue
333        *-------------------------------------------------------*/
334       GetNextChar(&c,cti);
335       continue;
336       }
337
338    }
339
340 return;
341 }
```

```
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*--------------------------------------------------------------------
 * tokenizer
 *--------------------------------------------------------------------*/

static int GetToken(
   CTokInfo    *cti
   )
   {
   int            c;
   int            type;
   unsigned long offsStart;

   /*-----------------------------------------------------------------
    * read next character
    *-----------------------------------------------------------------*/
   GetNextChar(&c,cti);

   /*-----------------------------------------------------------------
    * skip white space
    *-----------------------------------------------------------------*/
   while (isspace(c))
      GetNextChar(&c,cti);

   /*-----------------------------------------------------------------
    * save starting offset
    *-----------------------------------------------------------------*/
   offsStart = cti->fileOffs;

   /*-----------------------------------------------------------------
    * empty identifier
    *-----------------------------------------------------------------*/
   memset(cti->ident,'\0',sizeof(cti->ident));

   /*-----------------------------------------------------------------
    * big switch on it's value
    *-----------------------------------------------------------------*/
   switch(c)
      {
      /*--------------------------------------------------------------
       * check for end of file
       *--------------------------------------------------------------*/
      case EOF:
         type = TOKEN_EOF;
         break;

      /*--------------------------------------------------------------
       * for pound sign, read preprocessor directive
       *--------------------------------------------------------------*/
      case '#':
         ReadPreprocessor(cti);
         type = TOKEN_PREPROC;
         break;

      /*--------------------------------------------------------------
       * single or double quote
       *--------------------------------------------------------------*/
      case '\'':
      case '"':
         ReadString(cti,c);
         type = TOKEN_STRING;
         break;

      /*--------------------------------------------------------------
       * start of comment?
       *--------------------------------------------------------------*/
      case '/':
         /*-----------------------------------------------------------
          * get next char - if *, read to end of comment
          *-----------------------------------------------------------*/
         GetNextChar(&c,cti);
         if ('*' == c)
            {
            ReadComment(cti);
            type = TOKEN_COMMENT;
            }

         /*-----------------------------------------------------------
          * see if it's a C++ style comment
```

```
424             *-----------------------------------------------------------*/
425         else if ('/' == c)
426             {
427             ReadCppComment(cti);
428             type = TOKEN_COMMENT;
429             }
430 
431         /*-------------------------------------------------------
432          * otherwise it's just a plain /
433          *-----------------------------------------------------------*/
434         else
435             {
436             UnGetNextChar(c,cti);
437             type = TOKEN_OPER;
438             }
439 
440         break;
441 
442     /*-------------------------------------------------------------
443      * everything else - identifiers and punctuation
444      *-------------------------------------------------------------*/
445     default:
446         if (isCsymbol(c) && !isdigit(c))
447             {
448             ReadIdent(cti,c);
449             type = TOKEN_IDENT;
450             }
451 
452         else if (isdigit(c))
453             {
454             ReadNumber(cti,c);
455             type = TOKEN_NUMBER;
456             }
457 
458         /*-------------------------------------------------------
459          * anything else
460          *-------------------------------------------------------*/
461         else
462             {
463             type = TOKEN_OPER;
464             cti->ident[0] = (char) c;
465             }
466 
467         break;
468         }
469 
470     cti->tokOffs = offsStart;
471     cti->tokLen  = cti->fileOffs - offsStart + 1;
472     return(type);
473     }
```

```c

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/


/*------------------------------------------------------------------
 * Initializer
 *------------------------------------------------------------------*/
void *CTokInit(
   CTokRead  readFunc,
   void     *readInfo
   )
   {
   CTokInfo *cti;

   /*------------------------------------------------------------------
    * allocate space for structure
    *------------------------------------------------------------------*/
   cti = malloc(sizeof(CTokInfo));
   if (NULL == cti)
      return NULL;

   /*------------------------------------------------------------------
    * initialize structure
    *------------------------------------------------------------------*/
   cti->eof          = 0;
   cti->buffer       = NULL;
   cti->bufferLen    = 0L;
   cti->bufferInd    = 0L;
   cti->fileOffs     = -1L;
   cti->line         = 1;
   cti->unGetChar    = '\0';
   cti->unGetReady   = 0;
   cti->tokOffs      = 0L;
   cti->tokLen       = 0L;
   cti->readFunc     = readFunc;
   cti->readInfo     = readInfo;
   memset(cti->ident,'\0',sizeof(cti->ident));

   return cti;
   }

/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*------------------------------------------------------------------
 * Terminator
 *------------------------------------------------------------------*/
void CTokTerm(
   void *handle
   )
   {
   free(handle);
   }
```

```
528
529 /*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
530 /*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/
531
532 /*---------------------------------------------------------------------
533  * Tokenizer
534  *---------------------------------------------------------------------*/
535 void CTokGet(
536    void      *handle,
537    Token     *token
538    )
539    {
540    CTokInfo *cti;
541
542    cti = handle;
543
544    token->type  = GetToken(cti);
545    token->offs  = cti->tokOffs;
546    token->len   = cti->tokLen;
547    token->ident = cti->ident;
548    token->line  = cti->line;
549    }
```

```
1  /*-----------------------------------------------------------------
2   * hash.c : hash table functions
3   *-----------------------------------------------------------------
4   * 10-19-88 originally by Patrick J. Mueller
5   * 08-07-92 fixed up by Patrick J. Mueller
6   *-----------------------------------------------------------------*/
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 #include "list.h"
13 #include "hash.h"
```

```
14
15  /*-------------------------------------------------------------------
16   * create hash table
17   *-------------------------------------------------------------------*/
18  Hash *HashCreate(
19     int                 itemSize,
20     int                 buckets,
21     HashFunc            *hashFunc,
22     ListCompareFunc     *cmpFunc,
23     ListNoMemFunc       *memFunc
24     )
25     {
26     Hash *hash;
27     int   i;
28
29     /*-------------------------------------------------------------------
30      * sanity check
31      *-------------------------------------------------------------------*/
32     if (!itemSize || !buckets || !cmpFunc || !hashFunc)
33        return NULL;
34
35     /*-------------------------------------------------------------------
36      * allocate table structure
37      *-------------------------------------------------------------------*/
38     hash = malloc(sizeof(List));
39     if (!hash)
40        {
41        if (memFunc)
42           memFunc();
43        return NULL;
44        }
45
46     /*-------------------------------------------------------------------
47      * fill in fields
48      *-------------------------------------------------------------------*/
49     hash->itemSize = itemSize;
50     hash->buckets  = buckets;
51     hash->hashFunc = hashFunc;
52     hash->memFunc  = memFunc;
53
54     /*-------------------------------------------------------------------
55      * allocate buckets
56      *-------------------------------------------------------------------*/
57     hash->bucket = malloc(buckets*sizeof(List *));
58     if (!hash->bucket)
59        {
60        free(hash);
61
62        if (memFunc)
63           memFunc();
64
65        return NULL;
66        }
67
68     /*-------------------------------------------------------------------
69      * initialize to zero
70      *-------------------------------------------------------------------*/
71     memset(hash->bucket,0,buckets*sizeof(List *));
72
73     /*-------------------------------------------------------------------
74      * initialize buckets
75      *-------------------------------------------------------------------*/
76     for (i=0; i<buckets; i++)
77        {
78        hash->bucket[i] = ListCreate(itemSize,cmpFunc,memFunc);
79
80        if (!hash->bucket[i])
81           {
82           HashDestroy(hash);
83
84           if (memFunc)
85              memFunc();
86           }
87        }
88
89     /*-------------------------------------------------------------------
90      * return
91      *-------------------------------------------------------------------*/
92     return hash;
93     }
```

```
 94
 95  /*-------------------------------------------------------------------
 96   * destroy hash table
 97   *-------------------------------------------------------------------*/
 98  void HashDestroy(
 99     Hash *hash
100     )
101    {
102    int i;
103
104    if (!hash)
105       return;
106
107    for (i=0; i<hash->buckets; i++)
108       ListDestroy(hash->bucket[i]);
109
110    free(hash->bucket);
111    free(hash);
112    }
113
114  /*-------------------------------------------------------------------
115   * find entry in hash table
116   *-------------------------------------------------------------------*/
117  void *HashFind(
118     Hash *hash,
119     void *pItem
120     )
121    {
122    int h;
123
124    if (!hash)
125       return NULL;
126
127    h = hash->hashFunc(pItem,hash->buckets);
128    if ((h < 0) || (h >= hash->buckets))
129       return NULL;
130
131    return ListFind(hash->bucket[h],pItem);
132    }
133
134  /*-------------------------------------------------------------------
135   * add entry to hash table
136   *-------------------------------------------------------------------*/
137  void *HashAdd(
138     Hash *hash,
139     void *pItem
140     )
141    {
142    int h;
143
144    if (!hash)
145       return NULL;
146
147    h = hash->hashFunc(pItem,hash->buckets);
148    if ((h < 0) || (h >= hash->buckets))
149       return NULL;
150
151    return ListAdd(hash->bucket[h],pItem);
152    }
```

```
153
154  /*---------------------------------------------------------------------
155   * delete entry from hash table
156   *---------------------------------------------------------------------*/
157  void HashDelete(
158     Hash *hash,
159     void *pItem
160     )
161     {
162     int h;
163
164     if (!hash)
165        return;
166
167     h = hash->hashFunc(pItem,hash->buckets);
168     if ((h < 0) || (h >= hash->buckets))
169        return;
170
171     ListDelete(hash->bucket[h],pItem);
172     }
173
174  /*---------------------------------------------------------------------
175   * iterate through hash table
176   *---------------------------------------------------------------------*/
177  void HashIterate(
178     Hash            *hash,
179     ListIterateFunc *pIterateFunc,
180     void            *pUserData
181     )
182     {
183     int i;
184
185     if (!hash)
186        return;
187
188     for (i=0; i<hash->buckets; i++)
189        ListIterate(hash->bucket[i],pIterateFunc,pUserData);
190     }
191
192  /*---------------------------------------------------------------------
193   * test suite
194   *---------------------------------------------------------------------*/
195  #if defined(TEST)
196
197  /*---------------------------------------------------------------------
198   * compare function
199   *---------------------------------------------------------------------*/
200  static int compareFunc(
201     void *overi1,
202     void *overi2
203     )
204     {
205     int *i1 = overi1;
206     int *i2 = overi2;
207
208     if      (*i1 < *i2) return -1;
209     else if (*i1 > *i2) return  1;
210     else                return  0;
211     }
212
213  /*---------------------------------------------------------------------
214   * hash function
215   *---------------------------------------------------------------------*/
216  static int hashFunc(
217     void *overi,
218     int   buckets
219     )
220     {
221     int *i = overi;
222     return *i % buckets;
223     }
```

```
224
225 /*--------------------------------------------------------------------
226  * iterate function
227  *------------------------------------------------------------------*/
228 static void iterateFunc(
229    void *overI,
230    void *overCounter
231    )
232    {
233    int *pi       = overI;
234    int *pCounter = overCounter;
235
236    printf("%5d : %5d\n",*pCounter,*pi);
237    *pCounter += 1;
238    }
239
240 /*--------------------------------------------------------------------
241  *
242  *------------------------------------------------------------------*/
243 int main(void)
244    {
245    Hash *iHash;
246    int   i;
247    int   counter;
248
249    iHash = HashCreate(sizeof(int),3,compareFunc,hashFunc,NULL);
250
251    for (i= 1; i<10; i++)
252       HashAdd(iHash,&i);
253
254    for (i=20; i>10; i--)
255       HashAdd(iHash,&i);
256
257    for (i=0; i<=21; i++)
258       if (!HashFind(iHash,&i))
259          printf("didn't find %d\n",i);
260
261    counter = 1;
262    HashIterate(iHash,iterateFunc,&counter);
263
264    for (i=-1; i<5; i++)
265       HashDelete(iHash,&i);
266
267    for (i=21; i>15; i--)
268       HashDelete(iHash,&i);
269
270    counter = 1;
271    HashIterate(iHash,iterateFunc,&counter);
272
273    HashDestroy(iHash);
274
275    return 0;
276    }
277
278 #endif
```

```c
/*-------------------------------------------------------------------
 * list.c : list functions
 *-------------------------------------------------------------------
 * 10-19-88 originally by Patrick J. Mueller
 * 08-07-92 fixed up by Patrick J. Mueller
 *-------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "list.h"

/*-------------------------------------------------------------------
 * create a list
 *-------------------------------------------------------------------*/
List *ListCreate(
   int              itemSize,
   ListCompareFunc *cmpFunc,
   ListNoMemFunc   *memFunc
   )
{
List *list;

   /*-------------------------------------------------------------------
    * sanity check
    *-------------------------------------------------------------------*/
   if (!itemSize || !cmpFunc)
      return NULL;

   /*-------------------------------------------------------------------
    * allocate structure
    *-------------------------------------------------------------------*/
   list = malloc(sizeof(List));
   if (!list)
      {
      if (memFunc)
         memFunc();

      return NULL;
      }

   /*-------------------------------------------------------------------
    * set fields
    *-------------------------------------------------------------------*/
   list->head     = NULL;
   list->itemSize = itemSize;
   list->count    = 0;
   list->cmpFunc  = cmpFunc;
   list->memFunc  = memFunc;

   return list;
   }
```

```
54
55  /*-------------------------------------------------------------------
56   * destroy a list
57   *-------------------------------------------------------------------*/
58  void ListDestroy(
59     List *list
60     )
61     {
62     ListNode *node;
63     ListNode *next;
64
65     if (!list)
66        return;
67
68     /*----------------------------------------------------------------
69      * destroy each node
70      *----------------------------------------------------------------*/
71     node = list->head;
72     while (node)
73        {
74        next = node->next;
75        free(node->pItem);
76        free(node);
77        node = next;
78        }
79
80     /*----------------------------------------------------------------
81      * destroy list
82      *----------------------------------------------------------------*/
83     free(list);
84     }
85
86  /*-------------------------------------------------------------------
87   * get number of items in list
88   *-------------------------------------------------------------------*/
89  int ListCount(
90     List *list
91     )
92     {
93     if (!list)
94        return 0;
95     else
96        return list->count;
97     }
```

```
 98
 99  /*---------------------------------------------------------------
100   * find an item
101   *--------------------------------------------------------------*/
102  void *ListFind(
103     List *list,
104     void *pItem
105     )
106     {
107     ListNode *node;
108     int       cmp;
109
110     if (!list || !pItem)
111        return NULL;
112
113     /*---------------------------------------------------------------
114      * look for item
115      *--------------------------------------------------------------*/
116     for (node=list->head; node; node=node->next)
117        {
118        cmp = list->cmpFunc(pItem,node->pItem);
119
120        if (0 == cmp)
121           return node->pItem;
122
123        else if (cmp < 0)
124           return NULL;
125        }
126
127     return NULL;
128     }
```

```
129
130  /*---------------------------------------------------------------------
131   * add an item
132   *---------------------------------------------------------------------*/
133  void *ListAdd(
134     List       *list,
135     void       *pItem
136     )
137     {
138     ListNode *last;
139     ListNode *node;
140     ListNode *new;
141     int       cmp;
142
143     if (!list || !pItem)
144        return NULL;
145
146     /*---------------------------------------------------------------------
147      * find insertion point
148      *---------------------------------------------------------------------*/
149     last = NULL;
150     for (node=list->head; node; node=node->next)
151        {
152        cmp = (list->cmpFunc)(pItem,node->pItem);
153
154        if (0 == cmp)
155           return NULL;
156
157        else if (cmp < 0)
158           break;
159
160        last = node;
161        }
162
163     /*---------------------------------------------------------------------
164      * allocate memory
165      *---------------------------------------------------------------------*/
166     new = malloc(sizeof(ListNode));
167     if (new)
168        new->pItem = malloc(list->itemSize);
169
170     if (!new || !new->pItem)
171        {
172        if (list->memFunc)
173           list->memFunc();
174        return NULL;
175        }
176
177     /*---------------------------------------------------------------------
178      * update count, copy item
179      *---------------------------------------------------------------------*/
180     list->count++;
181     memcpy(new->pItem,pItem,list->itemSize);
182
183     /*---------------------------------------------------------------------
184      * link into list
185      *---------------------------------------------------------------------*/
186     if (last)
187        {
188        new->next  = last->next;
189        last->next = new;
190        }
191
192     else
193        {
194        new->next  = list->head;
195        list->head = new;
196        }
197
198     return new->pItem;
199     }
```

```
200
201 /*---------------------------------------------------------------------
202  * delete an item
203  *---------------------------------------------------------------------*/
204 void ListDelete(
205    List *list,
206    void *pItem
207    )
208 {
209 ListNode *last;
210 ListNode *node;
211 int      cmp;
212
213 if (!list || !pItem)
214    return;
215
216 /*---------------------------------------------------------------------
217  * find node
218  *---------------------------------------------------------------------*/
219 last = NULL;
220 for (node=list->head; node; node=node->next)
221    {
222    cmp = (list->cmpFunc)(pItem,node->pItem);
223    if (0 == cmp)
224       break;
225
226    else if (cmp < 0)
227       return;
228
229    last = node;
230    }
231
232 /*---------------------------------------------------------------------
233  * if not found, exit
234  *---------------------------------------------------------------------*/
235 if (!node)
236    return;
237
238 /*---------------------------------------------------------------------
239  * unlink from list
240  *---------------------------------------------------------------------*/
241 if (last)
242    {
243    if (last->next)
244       last->next = last->next->next;
245    else
246       last->next = NULL;
247    }
248
249 else
250    list->head = node->next;
251
252 /*---------------------------------------------------------------------
253  * update count, destroy item
254  *---------------------------------------------------------------------*/
255 list->count--;
256
257 free(node->pItem);
258 free(node);
259 }
```

```
260
261 /*---------------------------------------------------------------------
262  * iterate through items
263  *---------------------------------------------------------------------*/
264 void ListIterate(
265    List            *list,
266    ListIterateFunc *pIterateFunc,
267    void            *pUserData
268    )
269    {
270    ListNode *node;
271
272    if (!list || !pIterateFunc)
273       return;
274
275    for (node=list->head; node; node=node->next)
276       pIterateFunc(node->pItem,pUserData);
277    }
278
279 /*---------------------------------------------------------------------
280  * test suite
281  *---------------------------------------------------------------------*/
282 #if defined(TEST)
283
284 /*---------------------------------------------------------------------
285  * compare function
286  *---------------------------------------------------------------------*/
287 static int compareFunc(
288    void *overi1,
289    void *overi2
290    )
291    {
292    int *i1 = overi1;
293    int *i2 = overi2;
294
295    if      (*i1 < *i2) return -1;
296    else if (*i1 > *i2) return  1;
297    else                return  0;
298    }
299
300 /*---------------------------------------------------------------------
301  * iterate function
302  *---------------------------------------------------------------------*/
303 static void iterateFunc(
304    void *overI,
305    void *overCounter
306    )
307    {
308    int *pi       = overI;
309    int *pCounter = overCounter;
310
311    printf("%5d : %5d\n",*pCounter,*pi);
312    *pCounter += 1;
313    }
```

```
314
315 /*-------------------------------------------------------------------
316  *
317  *-------------------------------------------------------------------*/
318 int main(void)
319   {
320   List *iList;
321   int   i;
322   int   counter;
323
324   iList = ListCreate(sizeof(int),compareFunc,NULL);
325
326   printf("%d items\n",ListCount(iList));
327
328   for (i= 1; i<10; i++)
329       ListAdd(iList,&i);
330
331   for (i=20; i>10; i--)
332       ListAdd(iList,&i);
333
334   for (i=0; i<=21; i++)
335       if (!ListFind(iList,&i))
336           printf("didn't find %d\n",i);
337
338   printf("\n");
339   printf("%d items\n",ListCount(iList));
340   counter = 1;
341   ListIterate(iList,iterateFunc,&counter);
342
343   for (i=-1; i<5; i++)
344       ListDelete(iList,&i);
345
346   for (i=21; i>15; i--)
347       ListDelete(iList,&i);
348
349   printf("\n");
350   printf("%d items\n",ListCount(iList));
351   counter = 1;
352   ListIterate(iList,iterateFunc,&counter);
353
354   ListDestroy(iList);
355
356   return 0;
357   }
358
359 #endif
```

```c
/*--------------------------------------------------------------------*/
/* parsearg : parse parameters and options in an argv list            */
/*            see parsearg.h for a description                        */
/*--------------------------------------------------------------------*/
/* 03-13-90 originally by Patrick J. Mueller                          */
/* 01-09-91 version 2.0 by Patrick J. Mueller                         */
/* 04-29-91 version 3.0 by Patrick J. Mueller                         */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>

/*--------------------------------------------------------------------*/
/* typedefs                                                           */
/*--------------------------------------------------------------------*/
typedef enum
  {
  Boolean_Switch,
  Variable_Switch
  } Item_Type;

typedef struct Cmdline_Item
  {
  Item_Type             type;
  int                   position;
  char                  sw_char;
  void                 *variable;
  struct Cmdline_Item  *next;
  } Cmdline_Item;
```

```c
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-----------------------------------------------------------------*/
/*               L O C A L   F U N C T I O N S                     */
/*-----------------------------------------------------------------*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------*/
/* Search the Cmdline_Item list for a particular switch.  Look for  */
/* the option with a particular switch character (Boolean and       */
/* Variable are treated as the same)                                */
/*-----------------------------------------------------------------*/

static Cmdline_Item *get_item(
    Cmdline_Item *head,
    char          sw_char,
    int           case_sense
    )

    {
    Cmdline_Item *next;

    /*-------------------------------------------------------------*/
    /* traverse the linked list ...                                 */
    /*-------------------------------------------------------------*/
    next = head;
    while (next != NULL)
        {
        /*---------------------------------------------------------*/
        /* for case sensitive switches, just compare chars          */
        /*---------------------------------------------------------*/
        if (case_sense)
            {
            if (next->sw_char == sw_char)
                return(next);
            }

        /*---------------------------------------------------------*/
        /* otherwise, toupper the chars and compare                 */
        /*---------------------------------------------------------*/
        else
            {
            if (toupper(sw_char) == toupper(next->sw_char))
                return(next);
            }

        /*---------------------------------------------------------*/
        /* no matches so traverse to next item                      */
        /*---------------------------------------------------------*/
        next = next->next;
        }

    /*-------------------------------------------------------------*/
    /* no matches at all!                                           */
    /*-------------------------------------------------------------*/
    return(NULL);
    }
```

```c
/*-/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\-*/
/*-----------------------------------------------------------------*/
/*               M A I N     F U N C T I O N                       */
/*-----------------------------------------------------------------*/
/*-\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/-*/

/*-----------------------------------------------------------------*/
/* the main function                                               */
/*-----------------------------------------------------------------*/
void parsearg(
   int     *argc,
   char   **argv,
   int      case_sense,
   char    *env_var,
   char    *delimiters,
   char    *format_string,
   ...
   )

   {
   char         *tok;
   Cmdline_Item *item_head;
   Cmdline_Item *item_tail;
   Cmdline_Item *item;
   va_list       arg_marker;
   int           i;
   int           parms;
   int          *ptr_int;
   char        **ptr_ptr_char;
   char          sw_char;
   char         *env_value;
   int           envc;
   char         *envv;
   char         *temp;

   /*--------------------------------------------------------------*/
   /* sanity checks                                                */
   /*--------------------------------------------------------------*/
   if ((NULL == format_string)  ||
       (NULL == argv)           ||
       (0    == *argc))
      return;

   /*--------------------------------------------------------------*/
   /* make a copy of the format string since we will be strtok()ing */
   /* through it                                                   */
   /*--------------------------------------------------------------*/
   temp = malloc(1+strlen(format_string));
   if (NULL == temp)
      {
      puts("Error allocating memory in parsearg()");
      return;
      }

   strcpy(temp,format_string);
   format_string = temp;

   /*--------------------------------------------------------------*/
   /* get environment variable value                               */
   /*--------------------------------------------------------------*/
   env_value = NULL;

   if (NULL != env_var)
      if ('\0' != *env_var)
         {

         /*--------------------------------------------------------*/
         /* get value and copy if we found something               */
         /*--------------------------------------------------------*/
         env_value = getenv(env_var);

         if (NULL != env_value)
            {
            temp = malloc(1+strlen(env_value));
            if (NULL == temp)
               {
               puts("Error allocating memory in parsearg()");
               return;
               }

            strcpy(temp,env_value);
```

```
172             env_value = temp;
173             }
174         }
175
176     /*-------------------------------------------------------------*/
177     /* build option list                                           */
178     /*-------------------------------------------------------------*/
179     item_head = item_tail = NULL;
180     va_start(arg_marker,format_string);
181
182     /*-------------------------------------------------------------*/
183     /* parse the format_string with strtok                         */
184     /*-------------------------------------------------------------*/
185     tok = strtok(format_string," ");
186
187     while (NULL != tok)
188       {
189       /*-----------------------------------------------------------*/
190       /* allocate area for a new Item                              */
191       /*-----------------------------------------------------------*/
192       item = (Cmdline_Item *) malloc(sizeof(Cmdline_Item));
193       if (NULL == item)
194         {
195         puts("Error allocating memory in parsearg()");
196         return;
197         }
198
199       /*-----------------------------------------------------------*/
200       /* start assigning values to it                              */
201       /*-----------------------------------------------------------*/
202       item->next    = NULL;
203       item->sw_char = *tok++;
204
205       /*-----------------------------------------------------------*/
206       /* is it a boolean switch?                                   */
207       /*-----------------------------------------------------------*/
208       if ('\0' == *tok)
209          item->type = Boolean_Switch;
210
211       /*-----------------------------------------------------------*/
212       /* must be a variable switch                                 */
213       /*-----------------------------------------------------------*/
214       else
215          item->type = Variable_Switch;
216
217       /*-----------------------------------------------------------*/
218       /* now get the variable pointer                              */
219       /*-----------------------------------------------------------*/
220       item->variable = va_arg(arg_marker,void *);
221
222       /*-----------------------------------------------------------*/
223       /* initialize boolean switches to 0                          */
224       /*-----------------------------------------------------------*/
225       if (Boolean_Switch == item->type)
226         {
227         ptr_int  = item->variable;
228         *ptr_int = 0;
229         }
230
231       /*-----------------------------------------------------------*/
232       /* and variable switches to NULL                             */
233       /*-----------------------------------------------------------*/
234       else
235         {
236         ptr_ptr_char  = item->variable;
237         *ptr_ptr_char = NULL;
238         }
239
240       /*-----------------------------------------------------------*/
241       /* now insert into list (at end)                             */
242       /*-----------------------------------------------------------*/
243       if (NULL == item_head)
244          item_head = item_tail = item;
245
246       else
247         {
248         item_tail->next = item;
249         item_tail       = item;
250         }
251
252       /*-----------------------------------------------------------*/
253       /* get next item in format_string                            */
254       /*-----------------------------------------------------------*/
```

```
255         tok = strtok(NULL," ");
256       }
257
258     /*-------------------------------------------------------------*/
259     /* now we've set up the format_string.  time to step through the */
260     /* args to set the switches on and off and to scanf through    */
261     /* variable switches and parameters                            */
262     /*-------------------------------------------------------------*/
263
264     /*-------------------------------------------------------------*/
265     /* assign argc to parms and initialize argc to 0               */
266     /*-------------------------------------------------------------*/
267     parms = *argc;
268     *argc = 0;
269
270     /*-------------------------------------------------------------*/
271     /* We want to check the environment variables first.  Try to get */
272     /* the first item with strtok (if we had anything to begin with. */
273     /* If we have anything, set envv to the value and set env to 1.  */
274     /*-------------------------------------------------------------*/
275     envc = 0;
276     envv = NULL;
277
278     if (NULL != env_value)
279       {
280       envv = strtok(env_value," ");
281       if (NULL != envv)
282          envc = 1;
283       }
284
285     /*-------------------------------------------------------------*/
286     /* now loop through the environment variables and arguments    */
287     /* setting the appropriate value in the CmdLine_Item list.     */
288     /*-------------------------------------------------------------*/
289     for (i = 0; i < envc + parms; i++)
290       {
291       if (NULL != envv)
292          tok = envv;
293
294       else
295          tok = argv[i-envc];
296
297       /*----------------------------------------------------------*/
298       /* if it's a parameter, assign it to next argv pointer      */
299       /*----------------------------------------------------------*/
300       if (('\0' == *tok) || (NULL == strchr(delimiters,*tok)))
301         {
302
303         /*--------------------------------------------------------*/
304         /* we don't want to handle environment values though      */
305         /*--------------------------------------------------------*/
306         if (NULL == envv)
307           {
308           argv[*argc] = tok;
309           (*argc)++;
310           }
311         }
312
313       /*----------------------------------------------------------*/
314       /* otherwise it's a switch                                  */
315       /*----------------------------------------------------------*/
316       else
317         {
318         tok++;
319
320   parse_switches:
321         sw_char = *tok++;
322
323         /*--------------------------------------------------------*/
324         /* is it a switch?                                        */
325         /*--------------------------------------------------------*/
326         item = get_item(item_head,sw_char,case_sense);
327
328         if (NULL != item)
329           {
330           /*------------------------------------------------------*/
331           /* it's a switch, but is it variable or boolean?        */
332           /*------------------------------------------------------*/
333           if (Variable_Switch == item->type)
334             {
335             ptr_ptr_char  = item->variable;
336             *ptr_ptr_char = tok;
337             }
```

```
338
339                 else
340                 {
341                 ptr_int  = item->variable;
342                 *ptr_int = 1;
343
344                 /*-------------------------------------------------------*/
345                 /* handle multiple switches concatenated                 */
346                 /*-------------------------------------------------------*/
347                 if ('\0' != *tok)
348                    goto parse_switches;
349                 }
350             }
351
352         }
353
354         /*---------------------------------------------------------------*/
355         /* now get the next environment value if we need to              */
356         /*---------------------------------------------------------------*/
357         if (NULL != envv)
358           {
359           envv = strtok(NULL," ");
360           if (NULL != envv)
361              envc++;
362           }
363
364       }
365
366     /*-------------------------------------------------------------------*/
367     /* now release all the memory we used                                */
368     /*-------------------------------------------------------------------*/
369     item = item_head;
370     while (NULL != item)
371       {
372       item_head = item->next;
373       free(item);
374       item = item_head;
375       }
376
377     free(format_string);
378
379     /*-------------------------------------------------------------------*/
380     /* don't release this memory as we may have set pointers into it */
381     /* I guess if we wanted to be >real< tricky we could somehow     */
382     /* return pointers into the original string ... nah!!!           */
383     /*-------------------------------------------------------------------*/
384 #if defined(SHOOT_YOURSELF_IN_THE_FOOT)
385     if (NULL != env_value)
386        free(env_value);
387 #endif
388
389     return;
390     }
```

```c
/*-------------------------------------------------------------------
 * tokfile.c :
 *-------------------------------------------------------------------
 * 02-13-93 originally by Patrick J. Mueller
 *-------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "tokfile.h"

/*-------------------------------------------------------------------
 * types
 *-------------------------------------------------------------------*/
typedef struct
  {
  FILE *fHandle;
  int   index;
  int   length;
  char  line[TOKFILE_MAX_LINE+1];
  } TokFileData;

/*-------------------------------------------------------------------
 * open a file to be tokenized
 *-------------------------------------------------------------------*/
TokFileInfo TokFileOpen(
   char *fileName
   )
  {
  TokFileData *tfd;

  /*-----------------------------------------------------------------
   * allocate memory for handle
   *-----------------------------------------------------------------*/
  tfd = malloc(sizeof(TokFileData));
  if (!tfd)
     return NULL;

  memset(tfd,0,sizeof(TokFileData));

  /*-----------------------------------------------------------------
   * open the file
   *-----------------------------------------------------------------*/
  if (!strcmp(fileName,"-"))
     tfd->fHandle = stdin;
  else
     tfd->fHandle = fopen(fileName,"r");

  if (!tfd->fHandle)
     {
     free(tfd);
     return NULL;
     }

  /*-----------------------------------------------------------------
   * return handle
   *-----------------------------------------------------------------*/
  return (TokFileInfo) tfd;
  }
```

```
61
62  #define WHITE_SPACE " \a\b\f\n\r\t\v"
63
64  /*---------------------------------------------------------------
65   * get the next token from the file
66   *--------------------------------------------------------------*/
67  char *TokFileNext(
68     TokFileInfo tfi
69     )
70  {
71  TokFileData *tfd = tfi;
72  char        *tok;
73
74  if (!tfd)
75     return NULL;
76
77  if (!tfd->fHandle)
78     return NULL;
79
80  /*---------------------------------------------------------------
81   * loop trying to get the next token
82   *--------------------------------------------------------------*/
83  while (1)
84    {
85
86    /*-------------------------------------------------------------
87     * do we need to get another line
88     *-------------------------------------------------------------*/
89    while (tfd->index >= tfd->length)
90      {
91      fgets(tfd->line,TOKFILE_MAX_LINE,tfd->fHandle);
92
93      if (feof(tfd->fHandle))
94        {
95        fclose(tfd->fHandle);
96        free(tfd);
97        return NULL;
98        }
99
100     tfd->length = strlen(tfd->line);
101     tfd->index  = strspn(tfd->line,WHITE_SPACE);
102
103     /*----------------------------------------------------------
104      * check for comment
105      *----------------------------------------------------------*/
106     if ((tfd->index < tfd->length) && ('#' == tfd->line[tfd->index]))
107        tfd->index = tfd->length;
108     }
109
110   /*-------------------------------------------------------------
111    * skip past white space
112    *-------------------------------------------------------------*/
113   tfd->index += strspn(tfd->line + tfd->index, WHITE_SPACE);
114
115   if (tfd->index >= tfd->length)
116      continue;
117
118   tok = tfd->line + tfd->index;
119
120   /*-------------------------------------------------------------
121    * skip past non-white space
122    *-------------------------------------------------------------*/
123   tfd->index += strcspn(tfd->line + tfd->index, WHITE_SPACE);
124
125   tfd->line[tfd->index] = 0;
126
127   tfd->index++;
128
129   return tok;
130   }
131
132 return NULL;
133 }
```

```
134
135 #define TESTER_NOT
136 #if defined(TESTER)
137
138 int main(void)
139   {
140   TokFileInfo  tfi;
141   char        *token;
142
143   tfi = TokFileOpen("-");
144
145   while (token = TokFileNext(tfi))
146     printf("%s\n",token);
147
148   return 0;
149   }
150
151 #endif
```