```
0001 //Only one must be uncommented:
0002 #define _WIN32_ENVIRONMENT_
0003 //#define _POSIX_ENVIRONMENT_
0004
0005 #include <stdlib.h>             /* malloc(), Info from GNU C          */
0006 #include <stdio.h>              /* standard input/output routines.    */
0007 #include <dirent.h>             /* readdir(), etc.                    */
0008         #include <time.h>
0009         #include <utime.h>
0010 #include <sys/stat.h>           /* stat(), etc.                       */
0011 #include <unistd.h>             /* getcwd(), etc.                     */
0012
0013 #define MAX_DIR_PATH 2048        /* maximal full path we support.      */
0014
0015 #define UCHAR_MAX       255
0016
0017 #define INT_MAX                  2147483647
0018 #define INT_MIN                  (-INT_MAX-1)
0019 #define UINT_MAX        0xffffffff
0020
0021 #define SHRT_MAX        32767
0022 #define SHRT_MIN        (-SHRT_MAX-1)
0023 #define USHRT_MAX       0xffff
0024
0025 /*
0026  * Maximum and minimum values for longs and unsigned longs.
0027  *
0028  * TODO: This is not correct for Alphas, which have 64 bit longs.
0029  */
0030 #define LONG_MAX        2147483647L
0031 #define LONG_MIN        (-LONG_MAX-1)
0032 #define ULONG_MAX       0xffffffffUL
0033
0034 #define LONG_LONG_MAX   9223372036854775807LL
0035 #define LONG_LONG_MIN   (-LONG_LONG_MAX-1)
0036 #define ULONG_LONG_MAX  (2ULL * LONG_LONG_MAX + 1)
0037
0038 #ifndef NULL
0039 #ifdef __cplusplus
0040 #define NULL 0
0041 #else
0042 #define NULL ((void*)0)
0043 #endif
0044 #endif
0045
0046 #ifndef FALSE
0047 #define FALSE 0
0048 #endif
0049 #ifndef TRUE
0050 #define TRUE 1
0051 #endif
0052
0053 #ifndef false
0054 #define false 0
0055 #endif
0056 #ifndef true
0057 #define true 1
0058 #endif
0059
0060 #ifndef MAX_PATH
0061 #define       MAX_PATH  (260)
0062 #endif
0063 #define       _MAX_DIR  256
0064 #define       _MAX_FNAME          256
0065 #define       _MAX_EXT  256
0066
0067 typedef unsigned short  Bool;           /*  Boolean TRUE/FALSE value         */
0068 typedef unsigned short  bool;           /*  Boolean TRUE/FALSE value         */
0069 typedef unsigned char   byte;           /*  Single unsigned byte = 8 bits    */
0070
0071 #define FOREVER         for (;;)            /*  FOREVER { ... }              */
0072 #define until(expr)     while (!(expr))     /*  do { ... } until (expr)      */
0073 #define streq(s1,s2)    (!strcmp ((s1), (s2)))
0074 #define strneq(s1,s2)   (strcmp ((s1), (s2)))
0075 #define strused(s)      (*(s) != 0)
0076 #define strnull(s)      (*(s) == 0)
0077 #define strclr(s)       (*(s) = 0)
0078 #define strlast(s)      ((s) [strlen (s) - 1])
0079 #define strterm(s)      ((s) [strlen (s)])
0080
0081 #define KAZE_tolower(c)   ( (((c) >= 'A') && ((c) <= 'Z')) ? ((c) - 'A' + 'a') : (c) )
0082 #define KAZE_toupper(c)   ( (((c) >= 'a') && ((c) <= 'z')) ? ((c) - 'a' + 'A') : (c) )
0083 #define KAZE_iswalpha(c)  ( ('A' <= (c) && (c) <= 'Z') || ( 'a' <= (c) && (c) <= 'z'))
0084 #define KAZE_iswdigit(c)  ( '0' <= (c) && (c) <= '9')
0085
0086 #define KAZE_isascii(_c)  ( (unsigned)(_c) < 0x80 )
0087
0088 #define KAZE_max(a,b)     (((a) > (b)) ? (a) : (b))
```

```
0089 #define KAZE_min(a,b)     (((a) < (b)) ? (a) : (b))
0090
0091 #define       EXIT_SUCCESS       0
0092 #define       EXIT_FAILURE       1
0093
0094 long KAZE_strlen (
0095         const char * str
0096         )
0097 {
0098         const char *eos = str;
0099
0100         while( *eos++ ) ;
0101
0102         return( (int)(eos - str - 1) );
0103 }
0104
0105 // ----------------------------------------
0106 // this function tests is name matches mask
0107 // ? - any wchar_t or empty
0108 // * - any characters or empty
0109 static bool EnhancedMaskTest_OrEmpty(const char *mask, int maskPos,
0110                                      const char *name, int namePos)
0111 {
0112   int maskLen = KAZE_strlen(mask) - maskPos;
0113   int nameLen = KAZE_strlen(name) - namePos;
0114   if (maskLen == 0)
0115     if (nameLen == 0)
0116       return true;
0117     else
0118       return false;
0119   char maskChar = mask[maskPos];
0120   if(maskChar == '?')
0121   {
0122     /*
0123     if (EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos))
0124       return true;
0125     */
0126     if (EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos)) // KAZE: THIS LINE DECIDES whether 'or empty' or 'and not empty'
0127       return true;                                                  //       uncommented is 'or empty'
0128     if (nameLen == 0)
0129       return false;
0130     return EnhancedMaskTest_OrEmpty(mask,  maskPos + 1, name, namePos + 1);
0131   }
0132   else if(maskChar == '*')
0133   {
0134     if (EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos))
0135       return true;
0136     if (nameLen == 0)
0137       return false;
0138     return EnhancedMaskTest_OrEmpty(mask, maskPos, name, namePos + 1);
0139   }
0140   else
0141   {
0142     char c = name[namePos];
0143     //if (maskChar != c)
0144       if (KAZE_toupper(maskChar) != KAZE_toupper(c))
0145         return false;
0146     return EnhancedMaskTest_OrEmpty(mask,  maskPos + 1, name, namePos + 1);
0147   }
0148 }
0149
0150 // ----------------------------------------
0151 // this function tests is name matches mask
0152 // ? - any wchar_t and not empty
0153 // * - any characters or empty
0154 static bool EnhancedMaskTest_AndNotEmpty(const char *mask, int maskPos,
0155                                          const char *name, int namePos)
0156 {
0157   int maskLen = KAZE_strlen(mask) - maskPos;
0158   int nameLen = KAZE_strlen(name) - namePos;
0159   if (maskLen == 0)
0160     if (nameLen == 0)
0161       return true;
0162     else
0163       return false;
0164   char maskChar = mask[maskPos];
0165   if(maskChar == '?')
0166   {
0167     /*
0168     if (EnhancedMaskTest_AndNotEmpty(mask, maskPos + 1, name, namePos)) // KAZE: THIS LINE DECIDES whether 'or empty' or 'and not empty'
0169       return true;                                                      //       commented is 'and not empty'
0170     */
0171     if (nameLen == 0)
0172       return false;
0173     return EnhancedMaskTest_AndNotEmpty(mask,  maskPos + 1, name, namePos + 1);
0174   }
0175   else if(maskChar == '*')
0176   {
```

```
0177     if (EnhancedMaskTest_AndNotEmpty(mask, maskPos + 1, name, namePos))
0178       return true;
0179     if (nameLen == 0)
0180       return false;
0181     return EnhancedMaskTest_AndNotEmpty(mask, maskPos, name, namePos + 1);
0182   }
0183   else
0184   {
0185     char c = name[namePos];
0186     //if (maskChar != c)
0187       if (KAZE_toupper(maskChar) != KAZE_toupper(c))
0188         return false;
0189     return EnhancedMaskTest_AndNotEmpty(mask,  maskPos + 1, name, namePos + 1);
0190   }
0191 }
0192 bool CompareWildCardWithName(const char *mask, const char *name, int orempty)
0193 {
0194   if (orempty != 0)
0195   return EnhancedMaskTest_OrEmpty(mask, 0, name, 0);
0196   else
0197   return EnhancedMaskTest_AndNotEmpty(mask, 0, name, 0);
0198 }
0199 // Above fragment(modified) is from wildcard.cpp from 7zip package.
0200
0201 char * KAZE_strrev (
0202         char * string
0203         )
0204 {
0205         char *start = string;
0206         char *left = string;
0207         char ch;
0208
0209         while (*string++)                /* find end of string */
0210                 ;
0211         string -= 2;
0212
0213         while (left < string)
0214         {
0215                 ch = *left;
0216                 *left++ = *string;
0217                 *string-- = ch;
0218         }
0219
0220         return(start);
0221 }
0222
0223 ///*  ----------------------------------------------------------------------[<]-
0224 //    Function: file_matches
0225 //
0226 //    Synopsis: Returns TRUE if the filename matches the pattern.  The pattern
0227 //    is a character string that can contain these 'wildcard' characters:
0228 //    ----------------------------------------------------------------------[>]-*/
0229 //// THIS FUNCTION WORKS ONLY FOR ONE '*' I.E. MANY '*' ARE NOT ALLOWED! |(
0230 //Bool
0231 //file_matches (
0232 //    char *filename,
0233 //    char *pattern)
0234 //{
0235 //    char
0236 //        *pattern_ptr,                  /*  Points to pattern               */
0237 //        *filename_ptr,                 /*  Points to filename              */
0238 //        *filename_ptrLA; // Look Ahead
0239 //
0240 //    filename_ptr = (char *) filename;   /*  Start comparing file name       */
0241 //    pattern_ptr  = (char *) pattern;    /*  Start comparing file name       */
0242 //
0243 //    FOREVER
0244 //      {
0245 //        /*  If we came to the end of the pattern and the filename, we have   */
0246 //        /*  successful match.                                                */
0247 //        if (*pattern_ptr == '\0' && *filename_ptr == '\0')
0248 //            return (TRUE);             /*  Have a match                    */
0249 //
0250 //        /*  Otherwise, end of either is a failed match                       */
0251 //        if (*pattern_ptr == '\0' || *filename_ptr == '\0')
0252 //            return (FALSE);            /*  Match failed                    */
0253 //
0254 //        /*  If the pattern character is '?', then we matched a char          */
0255 //        if (*pattern_ptr == '?'
0256 //        ||  KAZE_toupper (*pattern_ptr) == KAZE_toupper (*filename_ptr))
0257 //          {
0258 //            pattern_ptr++;
0259 //            filename_ptr++;
0260 //          }
0261 //        else
0262 //        /*  If we have a '*', match as much of the filename as we can        */
0263 //        if (*pattern_ptr == '*')
0264 //          {
```

```
0265 //            pattern_ptr++;              /*  Try to match following char      */
0266 //            while (*filename_ptr && KAZE_toupper (*filename_ptr) != KAZE_toupper (*pattern_ptr))
0267 //                filename_ptr++;
0268 //        if (KAZE_strlen(pattern_ptr) != KAZE_strlen(filename_ptr))  // I can afford this due to
0269 //          {                                                         // no more '*' AND '?' - any character and not empty
0270 //                filename_ptrLA = filename_ptr;
0271 //                while (*filename_ptrLA && KAZE_toupper(*filename_ptrLA) != KAZE_toupper(*pattern_ptr))
0272 //                       filename_ptrLA++;
0273 //                if (KAZE_toupper(*filename_ptrLA) == KAZE_toupper(*pattern_ptr))
0274 //                {       pattern_ptr--;                      // Points to '*' again
0275 //                       if (*filename_ptr) filename_ptr++;  // Points to char after '.' f.e.
0276 //
0277 //                } else
0278 //                return (FALSE);                                   // FALSE only if no such char exists: case Tufto.exe.zip '.'
0279 //          }
0280 //          }
0281 //        else
0282 //                return (FALSE);             /*  Match failed                    */
0283 //
0284 //      }
0285 //}
0286
0287 // BELOW DISASM IS TO ENSSURE ME FOR SAFETY OF (X && Y) WHERE Y WILL NOT BE EXECUTED UNLESS X IS TRUE.
0288
0289 //09 int main( int argc, char **argv ) {
0290 //10     char
0291 //11         *pattern_ptr = NULL,                  /*  Points to pattern               */
0292 //12         *filename_ptr = NULL;                 /*  Points to filename              */
0293 //13
0294 //14             while (*filename_ptr && *filename_ptr != *pattern_ptr)
0295 //15                 filename_ptr++;
0296 //16
0297 //17     return( 0 );
0298 //18 }
0299
0300 //_main   PROC NEAR
0301 //; File c:\program files\microsoft visual c++ toolkit 2003\yoshi.c
0302 //; Line 12
0303 //        xor     ecx, ecx
0304 //; Line 14
0305 //        mov     al, BYTE PTR [ecx]
0306 //        push    esi
0307 //        xor     esi, esi
0308 //        test    al, al
0309 //        je      SHORT $L562
0310 //        mov     dl, BYTE PTR [esi]
0311 //        npad    3
0312 //$L549:
0313 //        cmp     al, dl
0314 //        je      SHORT $L562
0315 //        mov     al, BYTE PTR [ecx+1]
0316 //; Line 15
0317 //        inc     ecx
0318 //        test    al, al
0319 //        jne     SHORT $L549
0320 //$L562:
0321 //; Line 17
0322 //        xor     eax, eax
0323 //        pop     esi
0324 //; Line 18
0325 //        ret     0
0326 //_main   ENDP
0327
0328 char *Logo[] = {
0329 "Yoshi(Filelist Creator), revision 6+, written by Svalqyatchx,",
0330 "in fact based on SWEEP.C from 'Open Watcom Project', thanks-thanks.",
0331 "",
0332 "Note1: So far, it works for current directory only.",
0333 "Note2: Default method is depth-first traversal;",
0334 "       may use pipe 'Yoshi|sort' for breadth-first_like traversal results.",
0335 "Note3: Make notice that '*.*'(extensionfull only) is not equal to '*'(all);",
0336 "       one disadvantage is an inability to list only extensionless filenames.",
0337 "Note4: Search is case-insensitive as-must.",
0338 "Note5: This revision allows multiple '*', and meaning of masks is:",
0339 "       '?' - any character AND NOT EMPTY(default, for OR EMPTY see option -e);",
0340 "       '*' - any character(s) or empty.",
0341 "Note6: What is a .LBL(LineByLine) file?",
0342 "       it is a bunch of GRAMMATICAL lines not mere LF or CRLF lines;",
0343 "       it contains not symbols under 32(except CR and LF) and above 127;",
0344 "       it contains not space symbol sequences.",
0345 "Note7: Since r.6+ size of files bigger than 4GB is correctly reported.",
0346 "Usage:",
0347 "      Yoshi [option(s)] [filename(s)]",
0348 "      option(s):",
0349 "         -v          i.e. verbose mode; output goes to console;",
0350 "         -f          i.e. fullpath mode for output;",
0351 "         -e          i.e. treat '?' as any character OR EMPTY;",
0352 "         -t          i.e. touch all encountered files;",
```

```
0353 "         -2          i.e. convert all encountered .TXT files to .LBL files;",
0354 "         -o<filename> i.e. output goes to file(in append mode).",
0355 "      filename(s):",
0356 "         Wildcards '*' and wildcards '?' are allowed i.e. \"str*.c??\";",
0357 "         default filename is '*'; DO NOT FORGET TO PUT",
0358 "         filename(s) WITH WILDCARD(S) INTO QUOTE MARKS!",
0359
0360 // DAMN IGNORANCE LEADS TO(fix for below problem is QUOTE MARKS):
0361 //"Note2: I am confused: 'Yoshi *.c' receives all(i.e.) *.c one-by-one in current",
0362 //"       folder as command line parameters instead of WYSIWYG?! That leads",
0363 //"       to lost wildcard(s) i.e. expanding them and nonsense-buggy results.",
0364 //"       So, to avoid this ugly-pseudo-bug, the current folder must contain not",
0365 //"       extensions you are looking(with wildcard(s)) for - GRMBL.",
0366
0367 "Examples:",
0368 "      Yoshi -v -f -oCaterpillar_NON.lst \"*.lbl\" \"*.txt\" \"*.htm\" \"*.html\"",
0369 "      Yoshi -f -oMyEbooks.txt \"*wiley*essential*.pdf\" \"*russian*.*htm\"",
0370 "",
0371     NULL
0372 };
0373
0374     int verbose = 0;          /* verbose mode (default = false) */
0375     int orempty = 0;
0376     int mfullpath = 0;
0377     int lbl = 0;
0378     int tch = 0;
0379     int moutput = 0;
0380     unsigned long long int TotalSize = 0;
0381     unsigned long int TotalFiles = 0, TotalFolders = 0;
0382     unsigned long number_of_successful_touches = 0;
0383     unsigned long int LBLFiles = 0;
0384
0385     struct utimbuf stamp;
0386
0387 //char    SaveDir[MAX_PATH];
0388 char    SaveDir[MAX_DIR_PATH];
0389
0390 int Options_levels = INT_MAX;
0391 unsigned Options_depthfirst = 1;
0392
0393 typedef struct dirstack {
0394     struct dirstack     *prev;
0395     char                name_len;
0396     char                name[_MAX_FNAME + _MAX_EXT];
0397 }       dirstack;
0398
0399 dirstack        *Stack = NULL;
0400 int             DoneFlag = 0;
0401
0402
0403 void SetDoneFlag()
0404 {
0405     DoneFlag = 1;
0406 }
0407
0408
0409 void *SafeMalloc( size_t n )
0410 {
0411     void *p = malloc( n );
0412     if( p == NULL ) {
0413         puts( "Out of memory!" );
0414         SetDoneFlag();
0415     }
0416     return( p );
0417 }
0418
0419
0420 static char     CurrPathBuff[MAX_DIR_PATH + 2];
0421
0422 char *CurrPath()
0423 {
0424     char        *p;
0425     dirstack    *stack;
0426
0427     if( Stack->prev == NULL )
0428         return( "." );
0429     p = CurrPathBuff + MAX_DIR_PATH;
0430     *--p = '\0';
0431     stack = Stack;
0432     for( ;; ) {
0433         p -= stack->name_len;
0434         memcpy( p, stack->name, stack->name_len );
0435         stack = stack->prev;
0436         if( stack->prev == NULL )
0437     break;
0438         *--p = '\\';
0439     }
0440     return( p );
```

```
0441 }
0442
0443
0444 char *StringCopy( char *dst, char *src )
0445 {
0446     while( ( *dst = *src ) ) {
0447         ++dst;
0448         ++src;
0449     }
0450     return( dst );
0451 }
0452
0453 //PUBLIC  _StringCopy
0454 //; Function compile flags: /Ogty
0455 //_TEXT   SEGMENT
0456 //_dst$ = 8                                             ; size = 4
0457 //_src$ = 12                                            ; size = 4
0458 //_StringCopy PROC NEAR
0459 //; File c:\program files\microsoft visual c++ toolkit 2003\sweep_me1_.c
0460 //; Line 7
0461 //        mov     edx, DWORD PTR _src$[esp-4]
0462 //        mov     cl, BYTE PTR [edx]
0463 //        test    cl, cl
0464 //        mov     eax, DWORD PTR _dst$[esp-4]
0465 //        mov     BYTE PTR [eax], cl
0466 //        je      SHORT $L546
0467 //$L545:
0468 //        mov     cl, BYTE PTR [edx+1]
0469 //; Line 8
0470 //        inc     eax
0471 //; Line 9
0472 //        inc     edx
0473 //        test    cl, cl
0474 //        mov     BYTE PTR [eax], cl
0475 //        jne     SHORT $L545
0476 //$L546:
0477 //; Line 12
0478 //        ret     0
0479 //_StringCopy ENDP
0480
0481
0482 // TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL
0483 void TXT2LBL(char *CurrentSolid)
0484 {
0485       FILE *fp_unSOLID, *fp_SOLID;
0486       char LBLname[_MAX_FNAME + _MAX_EXT];
0487       char LBLname2[_MAX_FNAME + _MAX_EXT];
0488       long size_unSOLID, k;
0489       unsigned char workbyte, PREVworkbyte, LASTwrite_workbyte, c32 = ' ', c13 = '\r', c10 = '\n', cdot = '.';
0490       int NoLeadSPACE = 1;
0491
0492 StringCopy( LBLname, CurrentSolid );
0493 LBLname[KAZE_strlen(LBLname)-1] = '1';
0494 LBLname[KAZE_strlen(LBLname)-2] = 'B';
0495 LBLname[KAZE_strlen(LBLname)-3] = 'L';
0496 if( ( fp_unSOLID = fopen( CurrentSolid, "rb" ) ) == NULL )
0497 { printf( "Yoshi: Can't open %s file.\n", CurrentSolid ); exit( 1 ); }
0498 if( ( fp_SOLID = fopen( LBLname, "wb" ) ) == NULL )
0499 { printf( "Yoshi: Can't open %s file.\n", LBLname ); exit( 1 ); }
0500   fseek( fp_unSOLID, 0L, SEEK_END );
0501   size_unSOLID = ftell( fp_unSOLID );
0502   fseek( fp_unSOLID, 0L, SEEK_SET );
0503         workbyte = 0x00;
0504         LASTwrite_workbyte = 0x00;
0505         for( k = 0; k < size_unSOLID; k++ )
0506 {
0507          PREVworkbyte = workbyte;
0508          fread( &workbyte, 1, 1, fp_unSOLID );
0509          if( workbyte != c13 )
0510          {
0511                 // a file can finish with no LF character!!!
0512                 if (k != size_unSOLID - 1)
0513                        if( workbyte != c10 )
0514                        {
0515                        if (workbyte < c32)
0516                                {
0517                                if (LASTwrite_workbyte != c32)
0518                                fwrite( &c32, 1, 1, fp_SOLID );
0519                                LASTwrite_workbyte = c32;
0520                                }
0521                        else if (workbyte >= 0x80)
0522                                {
0523                                if (LASTwrite_workbyte != c32)
0524                                fwrite( &c32, 1, 1, fp_SOLID );
0525                                LASTwrite_workbyte = c32;
0526                                }
0527                        else
0528                                {
```

```
0529                                if (LASTwrite_workbyte == c32 && workbyte == c32) {} else
0530                                fwrite( &workbyte, 1, 1, fp_SOLID );
0531                                LASTwrite_workbyte = workbyte;
0532                                }
0533                        }
0534                        else
0535                        {
0536                        if (LASTwrite_workbyte != c32)
0537                        fwrite( &c32, 1, 1, fp_SOLID );
0538                        LASTwrite_workbyte = c32;
0539                        }
0540                  else
0541                  {
0542                    if (LASTwrite_workbyte != '?' && LASTwrite_workbyte != '!' && LASTwrite_workbyte != '.') fwrite( &cdot, 1, 1, fp_SOLID );
0543                    fwrite( &c13, 1, 1, fp_SOLID );
0544                    fwrite( &c10, 1, 1, fp_SOLID );
0545                  }
0546          }
0547        } // k 'for'
0548 fclose(fp_SOLID);
0549 fclose(fp_unSOLID);
0550
0551 StringCopy( LBLname2, LBLname );
0552 LBLname2[KAZE_strlen(LBLname2)-1] = 'L';
0553 LBLname2[KAZE_strlen(LBLname2)-2] = 'B';
0554 LBLname2[KAZE_strlen(LBLname2)-3] = 'L';
0555 if( ( fp_unSOLID = fopen( LBLname, "rb" ) ) == NULL )
0556 { printf( "Yoshi: Can't open %s file.\n", LBLname ); exit( 1 ); }
0557 if( ( fp_SOLID = fopen( LBLname2, "wb" ) ) == NULL )
0558 { printf( "Yoshi: Can't open %s file.\n", LBLname2 ); exit( 1 ); }
0559   fseek( fp_unSOLID, 0L, SEEK_END );
0560   size_unSOLID = ftell( fp_unSOLID );
0561   fseek( fp_unSOLID, 0L, SEEK_SET );
0562         workbyte = 0x00;
0563         for( k = 0; k < size_unSOLID - 2; k++ ) // -2 due to surely CRLF ending from LB1
0564 {
0565          PREVworkbyte = workbyte;
0566          fread( &workbyte, 1, 1, fp_unSOLID );
0567          if(( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte == 0x22)
0568          {
0569            fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
0570            fwrite( &workbyte, 1, 1, fp_SOLID );
0571            fwrite( &c13, 1, 1, fp_SOLID );
0572            fwrite( &c10, 1, 1, fp_SOLID );
0573            NoLeadSPACE = 1;
0574          }
0575          else if(( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte == 0x27)
0576          {
0577            fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
0578            fwrite( &workbyte, 1, 1, fp_SOLID );
0579            fwrite( &c13, 1, 1, fp_SOLID );
0580            fwrite( &c10, 1, 1, fp_SOLID );
0581            NoLeadSPACE = 1;
0582          }
0583          else if(( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte == 0x20)
0584          {
0585            fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
0586            fwrite( &c13, 1, 1, fp_SOLID );
0587            fwrite( &c10, 1, 1, fp_SOLID );
0588            NoLeadSPACE = 1;
0589          }
0590          else if(( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte != 0x20)
0591          {
0592            fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
0593            fwrite( &c13, 1, 1, fp_SOLID );
0594            fwrite( &c10, 1, 1, fp_SOLID );
0595            NoLeadSPACE = 1;
0596            if( workbyte != '.' && workbyte != '?' && workbyte != '!' ) {
0597            fwrite( &workbyte, 1, 1, fp_SOLID );
0598            NoLeadSPACE = 0; }
0599            if (k == (size_unSOLID - 2) - 1)
0600            {
0601            fwrite( &workbyte, 1, 1, fp_SOLID ); // last char is .|?|!
0602            fwrite( &c13, 1, 1, fp_SOLID );
0603            fwrite( &c10, 1, 1, fp_SOLID );
0604            }
0605          }
0606          else
0607          {
0608            if( workbyte != '.' && workbyte != '?' && workbyte != '!' ) {
0609            if( workbyte == c32 && NoLeadSPACE == 1 ) {} else {
0610            fwrite( &workbyte, 1, 1, fp_SOLID );
0611            NoLeadSPACE = 0; }}
0612            if (k == (size_unSOLID - 2) - 1)
0613            {
0614            fwrite( &workbyte, 1, 1, fp_SOLID ); // last char is .|?|!
0615            fwrite( &c13, 1, 1, fp_SOLID );
0616            fwrite( &c10, 1, 1, fp_SOLID );
```

```
0617            }
0618          }
0619        } // k 'for'
0620 fclose(fp_SOLID);
0621 fclose(fp_unSOLID);
0622
0623 if( remove( LBLname ) != 0 )
0624 { printf( "Yoshi: Can't remove %s file.\n", LBLname ); exit( 1 ); }
0625 }
0626 // TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL
0627
0628 // Again thanks to TOUCH.C from WATCOM:
0629 static int doTouchFile( char *full_name, struct utimbuf *stamp)
0630 /*********************************************/
0631 {
0632     int utime_rc;
0633
0634     utime_rc = utime( full_name, stamp );
0635     if( utime_rc == -1 ) {
0636     } else {
0637         return 1;
0638     }
0639     return 0;
0640 }
0641
0642 void ProcessCurrentDirectory(char *WildCard, FILE *fp_outLOG)
0643 {
0644     DIR                 *dirh;
0645     struct dirent       *dp;
0646     dirstack            *stack;
0647     char cwd[MAX_DIR_PATH+1];    /* current working directory.        */
0648
0649     if( !Options_depthfirst ) {
0650         //ExecuteCommands();
0651     }
0652     if( Options_levels != 0 ) {
0653         dirh = opendir( "." );
0654         if( dirh != NULL ) {
0655             --Options_levels;
0656             for( ;; ) {
0657                 if( DoneFlag )
0658          return;
0659                 dp = readdir( dirh );
0660                 if( dp == NULL )
0661          break;
0662 //#ifdef __UNIX__
0663                 {
0664 #if defined(_WIN32_ENVIRONMENT_)
0665                     struct _stati64 buf; // this is new: r.6+
0666 #else
0667                     struct stat buf; // this is old: r.6; Linux does what it should: this works for 64bit automatically
0668 #endif /* defined(_WIN32_ENVIRONMENT_)  */
0669 #if defined(_WIN32_ENVIRONMENT_)
0670                     if ( _stati64( dp->d_name, &buf ) != 0 )
0671 #else
0672                     if ( stat( dp->d_name, &buf ) != 0 ) // this is old: r.6
0673 #endif /* defined(_WIN32_ENVIRONMENT_)  */
0674                     { // ######### CRASH SPOT ???? !!!!! #########
0675                       // my case was for folders that are bad i.e. cannot be erase or copy
0676                       printf("Yoshi: Break after function 'stat'.\n");
0677                       exit( EXIT_FAILURE ); }
0678                     if( !S_ISDIR( buf.st_mode ) )
0679                     {
0680 //if( S_ISREG( buf.st_mode ) && file_matches ( dp->d_name, WildCard ) ) // Under Windows it reads hidden and system files - grmbl.
0681 if( S_ISREG( buf.st_mode ) && CompareWildCardWithName ( WildCard, dp->d_name, orempty ) )
0682 {
0683 TotalSize = TotalSize + buf.st_size;
0684 TotalFiles++;
0685
0686                        if (tch != 0) {
0687                           number_of_successful_touches +=
0688                           doTouchFile( dp->d_name, &stamp );
0689                        }
0690
0691         if (lbl != 0) {
0692          KAZE_strrev(dp->d_name);
0693          if (KAZE_toupper(dp->d_name[0]) == 'T' && KAZE_toupper(dp->d_name[1]) == 'X' && KAZE_toupper(dp->d_name[2]) == 'T' && dp->d_name[3]
     == '.') {
0694           KAZE_strrev(dp->d_name);
0695            LBLFiles++;
0696            printf("Converting(LBLing) %s ...\n", dp->d_name);
0697            TXT2LBL (dp->d_name);
0698           } else
0699           KAZE_strrev(dp->d_name);
0700          }
0701         if (mfullpath != 0) {
0702             if ( !getcwd(cwd, MAX_DIR_PATH+1) ) {
0703             // Some error
```

```
0704             } else {
0705                 if (moutput != 0) {
0706                 fprintf( fp_outLOG, "%s\\%s\n", cwd, dp->d_name );
0707                 }
0708                 if (verbose != 0) printf("%s\\%s\n", cwd, dp->d_name);
0709             }
0710         } else {
0711                 if (moutput != 0) {
0712                 fprintf( fp_outLOG, "%s\\%s\n", CurrPath(), dp->d_name );
0713                 }
0714                 if (verbose != 0) printf("%s\\%s\n", CurrPath(), dp->d_name);
0715         }
0716 }
0717             continue;
0718                 }
0719                 }
0720 //#else
0721 //                if( !( dp->d_attr & _A_SUBDIR ) )
0722 //                    continue;
0723 //#endif
0724                 if( dp->d_name[0] == '.' ) {
0725                     if( dp->d_name[1] == '.' || dp->d_name[1] == '\0' )
0726             continue;
0727                 }
0728                 stack = SafeMalloc( sizeof( *stack ) );
0729                 if( DoneFlag )
0730          return;
0731                 stack->name_len = strlen( dp->d_name );
0732                 memcpy( stack->name, dp->d_name, stack->name_len + 1 );
0733                 stack->prev = Stack;
0734                 Stack = stack;
0735
0736 TotalFolders++;
0737 //puts(CurrPath()); // All folders listing
0738
0739                 if (chdir( stack->name ) != 0)
0740                     { // ######### CRASH SPOT ???? !!!!! #########
0741                       // my case was for hidden folder System Volume Information
0742                       printf("Yoshi: Break after function 'chdir'.\n");
0743                       exit( EXIT_FAILURE ); }
0744                 ProcessCurrentDirectory( WildCard, fp_outLOG );
0745                 chdir( ".." );
0746                 Stack = stack->prev;
0747                 free( stack );
0748             }
0749             ++Options_levels;
0750             closedir( dirh );
0751         }
0752     }
0753     if( Options_depthfirst ) {
0754         //ExecuteCommands();
0755     }
0756 }
0757
0758 void PrintLogo()
0759 {
0760     int i;
0761
0762     for( i = 0; Logo[i] != NULL; ++i )
0763         puts( Logo[i] );
0764     //exit( EXIT_FAILURE );
0765 }
0766
0767 //int scmp( unsigned char *s1, unsigned char *s2 )
0768 //{
0769 //    while( *s1 != '\0' && *s1 == *s2 )
0770 //    {
0771 //        s1++;
0772 //        s2++;
0773 //    }
0774 //    return( *s1-*s2 );
0775 //}
0776
0777 void x64toaKAZE (      /* stdcall is faster and smaller... Might as well use it for the helper. */
0778         unsigned long long val,
0779         char *buf,
0780         unsigned radix,
0781         int is_neg
0782         )
0783 {
0784         char *p;                /* pointer to traverse string */
0785         char *firstdig;         /* pointer to first digit */
0786         char temp;              /* temp char */
0787         unsigned digval;        /* value of digit */
0788
0789         p = buf;
0790
0791         if ( is_neg )
```

```
0792         {
0793             *p++ = '-';         /* negative, so output '-' and negate */
0794             val = (unsigned long long)(-(long long)val);
0795         }
0796
0797         firstdig = p;           /* save pointer to first digit */
0798
0799         do {
0800             digval = (unsigned) (val % radix);
0801             val /= radix;       /* get next digit */
0802
0803             /* convert to ascii and store */
0804             if (digval > 9)
0805                 *p++ = (char) (digval - 10 + 'a');  /* a letter */
0806             else
0807                 *p++ = (char) (digval + '0');       /* a digit */
0808         } while (val > 0);
0809
0810         /* We now have the digit of the number in the buffer, but in reverse
0811            order.  Thus we reverse them now. */
0812
0813         *p-- = '\0';            /* terminate string; p points to last digit */
0814
0815         do {
0816             temp = *p;
0817             *p = *firstdig;
0818             *firstdig = temp;   /* swap *p and *firstdig */
0819             --p;
0820             ++firstdig;         /* advance to next two digits */
0821         } while (firstdig < p); /* repeat until halfway */
0822 }
0823
0824 char * _ui64toaKAZEzerocomma (
0825         unsigned long long val,
0826         char *buf,
0827         int radix
0828         )
0829 {
0830                         char *p;
0831                         char temp;
0832                         int txpman;
0833                         int pxnman;
0834         x64toaKAZE(val, buf, radix, 0);
0835                         p = buf;
0836                         do {
0837                         } while (*++p != '\0');
0838                         p--; // p points to last digit
0839                              // buf points to first digit
0840                         buf[26] = 0;
0841                         txpman = 1;
0842                         pxnman = 0;
0843                         do
0844                         { if (buf <= p)
0845                           { temp = *p;
0846                             buf[26-txpman] = temp; pxnman++;
0847                             p--;
0848                             if (pxnman % 3 == 0)
0849                             { txpman++;
0850                               buf[26-txpman] = (char) (',');
0851                             }
0852                           }
0853                           else
0854                           { buf[26-txpman] = (char) ('0'); pxnman++;
0855                             if (pxnman % 3 == 0)
0856                             { txpman++;
0857                               buf[26-txpman] = (char) (',');
0858                             }
0859                           }
0860                           txpman++;
0861                         } while (txpman <= 26);
0862         return buf;
0863 }
0864
0865 // Again thanks to TOUCH.C from WATCOM:
0866 static void syncStamp( struct utimbuf *stamp )
0867 /*****************************************/
0868 //struct tm
0869 //{
0870 //        int     tm_sec;         /* Seconds: 0-59 (K&R says 0-61?) */
0871 //        int     tm_min;         /* Minutes: 0-59 */
0872 //        int     tm_hour;        /* Hours since midnight: 0-23 */
0873 //        int     tm_mday;        /* Day of the month: 1-31 */
0874 //        int     tm_mon;         /* Months *since* january: 0-11 */
0875 //        int     tm_year;        /* Years since 1900 */
0876 //        int     tm_wday;        /* Days since Sunday (0-6) */
0877 //        int     tm_yday;        /* Days since Jan. 1: 0-365 */
0878 //        int     tm_isdst;       /* +1 Daylight Savings Time, 0 No DST,
0879 //                                 * -1 don't know */
```

```
0880 //};
0881 //_CRTIMP time_t __cdecl  mktime (struct tm*);
0882 //_CRTIMP struct tm*  __cdecl     localtime (const time_t*);
0883
0884 {
0885     time_t touch_time;
0886     static struct tm touchTime;
0887
0888     time_t      curr_time;
0889     struct tm   *ptime;
0890
0891     time( &curr_time );
0892     ptime = localtime( &curr_time );
0893     touchTime = *ptime;
0894
0895     //    touchTime.tm_sec = 0;
0896     //    touchTime.tm_min = 0;
0897     //    touchTime.tm_hour = 0;
0898         //touchTime.tm_mday = DateAdjust.day;
0899         //touchTime.tm_mon = DateAdjust.month - 1;
0900         //touchTime.tm_year = DateAdjust.year - 1900;
0901         //touchTime.tm_isdst = -1;
0902
0903     touch_time = mktime( &touchTime );
0904     stamp->actime  = touch_time;
0905     stamp->modtime = touch_time;
0906 }
0907
0908 int main( int argc, char **argv ) {
0909 //int main(int argc, char *argv[]) {
0910
0911     char llTOaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
0912     char *WildCard = "*";
0913     char *out_file = "Yoshi.lst";
0914     FILE *fp_outLOG;
0915
0916     PrintLogo();
0917     Stack = SafeMalloc( sizeof( *Stack ) );
0918     Stack->name_len = 1;
0919     Stack->prev = NULL;
0920     StringCopy( Stack->name, "." );
0921     getcwd( SaveDir, MAX_DIR_PATH+1 );
0922 //chdir( "d:\\www.sanmayce.com" );
0923
0924     /*
0925      * loop for each option.
0926      *   Stop if we run out of arguments
0927      *   or we get an argument without a dash.
0928      */
0929     while ((argc > 1) && (argv[1][0] == '-')) {
0930         /*
0931          * argv[1][1] is the actual option character.
0932          */
0933         switch (argv[1][1]) {
0934             /*
0935              * -v verbose
0936              */
0937             case 'v':
0938                 verbose = 1;
0939                 break;
0940             case 'e':
0941                 orempty = 1;
0942                 break;
0943             case '2':
0944                 lbl = 1;
0945                 break;
0946             case 't':
0947                 tch = 1;
0948                 syncStamp( &stamp );
0949                 break;
0950             case 'f':
0951                 mfullpath = 1;
0952                 break;
0953             /*
0954              * -o<name>  output file
0955              *    [0] is the dash
0956              *    [1] is the "o"
0957              *    [2] starts the name
0958              */
0959             case 'o':
0960                 moutput = 1;
0961                 out_file = &argv[1][2];
0962 if( ( fp_outLOG = fopen( out_file, "ab" ) ) == NULL )
0963 { printf( "Yoshi: Can't open file %s.\n", out_file ); return( EXIT_FAILURE ); }
0964
0965                 break;
0966             /*
0967              * -l<number> set max number of lines
```

```
0968              */
0969             //case 'l':
0970             //    line_max = atoi(&argv[1][2]);
0971             //    break;
0972             default:
0973                 printf("Yoshi: Bad option %s\n", argv[1]);
0974                 return( EXIT_FAILURE );
0975         }
0976         /*
0977          * move the argument list up one
0978          * move the count down one
0979          */
0980         ++argv;
0981         --argc;
0982     }
0983
0984     /*
0985      * At this point all the options have been processed.
0986      * Check to see if we have no files in the list
0987      * and if so, we need to process just standard in.
0988      */
0989         if (argc == 1) {
0990     ProcessCurrentDirectory( WildCard, fp_outLOG ); // *.* is default
0991         } else {
0992         while (argc > 1) {
0993         WildCard = argv[1];
0994     ProcessCurrentDirectory( WildCard, fp_outLOG );
0995           ++argv;
0996           --argc;
0997         }
0998         }
0999
1000 if ((verbose == 1) && TotalFiles)
1001 printf("\n");
1002 else
1003 if ((lbl == 1) && LBLFiles)
1004 printf("\n");
1005
1006 printf("Yoshi: Total size of files: %s bytes.\n", _ui64toaKAZEzerocomma(TotalSize, llTOaDigits, 10)+(26-18)) ; // 26 are all 26-DESIRED=24
1007 if (tch == 1)
1008 printf("Yoshi: Total files: %s touched.\n", _ui64toaKAZEzerocomma(number_of_successful_touches, llTOaDigits, 10)+(26-15)) ; // 26 are all 26-
     DESIRED=24
1009 printf("Yoshi: Total files: %s.\n", _ui64toaKAZEzerocomma(TotalFiles, llTOaDigits, 10)+(26-15)) ; // 26 are all 26-DESIRED=24
1010 printf("Yoshi: Total folders: %s.\n", _ui64toaKAZEzerocomma(TotalFolders, llTOaDigits, 10)+(26-13)) ; // 26 are all 26-DESIRED=24
1011
1012     free( Stack );
1013     Stack = NULL;
1014     chdir( SaveDir );
1015 if (moutput == 1) fclose( fp_outLOG );
1016     return( EXIT_SUCCESS );
1017 }
```