

```

#include <stdlib.h>          /* malloc(), Info from GNU C      */
#include <stdio.h>           /* standard input/output routines. */
#include <dirent.h>          /* readdir(), etc.                */
    #include <time.h>
    #include <utime.h>
#include <sys/stat.h>        /* stat(), etc.                  */
#include <unistd.h>          /* getcwd(), etc.                */

#define MAX_DIR_PATH 2048   /* maximal full path we support. */

#define UCHAR_MAX    255

#define INT_MAX        2147483647
#define INT_MIN        (-INT_MAX-1)
#define UINT_MAX       0xffffffff

#define SHRT_MAX       32767
#define SHRT_MIN       (-SHRT_MAX-1)
#define USHRT_MAX      0xffff

/*
 * Maximum and minimum values for longs and unsigned longs.
 *
 * TODO: This is not correct for Alphas, which have 64 bit longs.
 */
#define LONG_MAX        2147483647L
#define LONG_MIN        (-LONG_MAX-1)
#define ULONG_MAX       0xffffffffUL

#define LONG_LONG_MAX   9223372036854775807LL
#define LONG_LONG_MIN   (-LONG_LONG_MAX-1)
#define ULONG_LONG_MAX (2ULL * LONG_LONG_MAX + 1)

#ifndef NULL
#ifdef __cplusplus
#define NULL 0
#else
#define NULL ((void*)0)
#endif
#endif

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

#ifndef false
#define false 0
#endif
#ifndef true
#define true 1
#endif

#ifndef MAX_PATH
#define MAX_PATH        (260)
#endif
#define _MAX_DIR        256
#define _MAX_FNAME      256
#define _MAX_EXT        256

typedef unsigned short   Bool;          /* Boolean TRUE/FALSE value */
typedef unsigned short   bool;         /* Boolean TRUE/FALSE value */
typedef unsigned char    byte;         /* Single unsigned byte = 8 bits */

```

```

#define FOREVER      for (;;)          /* FOREVER { ... }          */
#define until(expr)  while (!(expr))   /* do { ... } until (expr) */
#define streq(s1,s2) (!strcmp ((s1), (s2)))
#define strneq(s1,s2) (strcmp ((s1), (s2)))
#define strused(s)   (*(s) != 0)
#define strnull(s)   (*(s) == 0)
#define strclr(s)    (*(s) = 0)
#define strlast(s)   ((s) [strlen (s) - 1])
#define strterm(s)   ((s) [strlen (s)])

#define KAZE_tolower(c)  ( (((c) >= 'A') && ((c) <= 'Z')) ? ((c) - 'A' + 'a') : (c) )
#define KAZE_toupper(c)  ( (((c) >= 'a') && ((c) <= 'z')) ? ((c) - 'a' + 'A') : (c) )
#define KAZE_iswalpha(c)  ( ('A' <= (c) && (c) <= 'Z') || ( 'a' <= (c) && (c) <= 'z') )
#define KAZE_iswdigit(c)  ( '0' <= (c) && (c) <= '9' )

#define KAZE_isascii(_c)  ( (unsigned)(_c) < 0x80 )

#define KAZE_max(a,b)     (((a) > (b)) ? (a) : (b))
#define KAZE_min(a,b)     (((a) < (b)) ? (a) : (b))

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1

long KAZE_strlen (
    const char * str
)
{
    const char *eos = str;

    while( *eos++ ) ;

    return( (int)(eos - str - 1) );
}

// -----
// this function tests is name matches mask
// ? - any wchar_t or empty
// * - any characters or empty
static bool EnhancedMaskTest_OrEmpty(const char *mask, int maskPos,
                                     const char *name, int namePos)
{
    int maskLen = KAZE_strlen(mask) - maskPos;
    int nameLen = KAZE_strlen(name) - namePos;
    if (maskLen == 0)
        if (nameLen == 0)
            return true;
        else
            return false;
    char maskChar = mask[maskPos];
    if(maskChar == '?')
    {
        /*
        if (EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos))
            return true;
        */
        if (EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos)) // KAZE: THIS LINE DECIDES
whether 'or empty' or 'and not empty'
            return true;
//          uncommented is 'or
empty'
        if (nameLen == 0)
            return false;
        return EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos + 1);
    }
    else if(maskChar == '*')

```

```

{
    if (EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos))
        return true;
    if (nameLen == 0)
        return false;
    return EnhancedMaskTest_OrEmpty(mask, maskPos, name, namePos + 1);
}
else
{
    char c = name[namePos];
    //if (maskChar != c)
    if (KAZE_toupper(maskChar) != KAZE_toupper(c))
        return false;
    return EnhancedMaskTest_OrEmpty(mask, maskPos + 1, name, namePos + 1);
}
}

```

```

// -----
// this function tests is name matches mask
// ? - any wchar_t and not empty
// * - any characters or empty

```

```

static bool EnhancedMaskTest_AndNotEmpty(const char *mask, int maskPos,
                                         const char *name, int namePos)

```

```

{
    int maskLen = KAZE_strlen(mask) - maskPos;
    int nameLen = KAZE_strlen(name) - namePos;
    if (maskLen == 0)
        if (nameLen == 0)
            return true;
        else
            return false;
    char maskChar = mask[maskPos];
    if (maskChar == '?')
    {
        /*
        if (EnhancedMaskTest_AndNotEmpty(mask, maskPos + 1, name, namePos)) // KAZE: THIS LINE DECIDES
whether 'or empty' or 'and not empty'
            return true;
not empty'
        */
        if (nameLen == 0)
            return false;
        return EnhancedMaskTest_AndNotEmpty(mask, maskPos + 1, name, namePos + 1);
    }
    else if (maskChar == '*')
    {
        if (EnhancedMaskTest_AndNotEmpty(mask, maskPos + 1, name, namePos))
            return true;
        if (nameLen == 0)
            return false;
        return EnhancedMaskTest_AndNotEmpty(mask, maskPos, name, namePos + 1);
    }
    else
    {
        char c = name[namePos];
        //if (maskChar != c)
        if (KAZE_toupper(maskChar) != KAZE_toupper(c))
            return false;
        return EnhancedMaskTest_AndNotEmpty(mask, maskPos + 1, name, namePos + 1);
    }
}

```

```

bool CompareWildCardWithName(const char *mask, const char *name, int orempty)

```

```

{
    if (orempty != 0)
        return EnhancedMaskTest_OrEmpty(mask, 0, name, 0);
}

```

```

else
    return EnhancedMaskTest_AndNotEmpty(mask, 0, name, 0);
}
// Above fragment(modified) is from wildcard.cpp from 7zip package.

```

```

char * KAZE_strrev (
    char * string
)
{
    char *start = string;
    char *left = string;
    char ch;

    while (*string++)          /* find end of string */
        ;
    string -= 2;

    while (left < string)
    {
        ch = *left;
        *left++ = *string;
        *string-- = ch;
    }

    return(start);
}

```

```

///<* -----[<]-
// Function: file_matches
//
// Synopsis: Returns TRUE if the filename matches the pattern. The pattern
// is a character string that can contain these 'wildcard' characters:
// -----[>]-*/
//// THIS FUNCTION WORKS ONLY FOR ONE '*' I.E. MANY '*' ARE NOT ALLOWED! |(
//Bool
//file_matches (
//    char *filename,
//    char *pattern)
//{
//    char
//        *pattern_ptr,          /* Points to pattern          */
//        *filename_ptr,         /* Points to filename     */
//        *filename_ptrLA; // Look Ahead
//
//    filename_ptr = (char *) filename; /* Start comparing file name */
//    pattern_ptr = (char *) pattern;   /* Start comparing file name */
//
//    FOREVER
//    {
//        /* If we came to the end of the pattern and the filename, we have
//        /* successful match.
//        if (*pattern_ptr == '\0' && *filename_ptr == '\0')
//            return (TRUE);          /* Have a match */
//
//        /* Otherwise, end of either is a failed match
//        if (*pattern_ptr == '\0' || *filename_ptr == '\0')
//            return (FALSE);         /* Match failed */
//
//        /* If the pattern character is '?', then we matched a char
//        if (*pattern_ptr == '?')
//            || KAZE_toupper (*pattern_ptr) == KAZE_toupper (*filename_ptr))
//            {
//                pattern_ptr++;
//                filename_ptr++;
//            }
//    }
//}

```

```

//      else
//      /* If we have a '*', match as much of the filename as we can */
//      if (*pattern_ptr == '*')
//      {
//          pattern_ptr++;
//          /* Try to match following char */
//          while (*filename_ptr && KAZE_toupper (*filename_ptr) != KAZE_toupper (*pattern_ptr))
//              filename_ptr++;
//          if (KAZE_strlen(pattern_ptr) != KAZE_strlen(filename_ptr)) // I can afford this due to
//              { // no more '*' AND '?' - any
character and not empty
//              filename_ptrLA = filename_ptr;
//              while (*filename_ptrLA && KAZE_toupper(*filename_ptrLA) !=
KAZE_toupper(*pattern_ptr))
//                  filename_ptrLA++;
//              if (KAZE_toupper(*filename_ptrLA) == KAZE_toupper(*pattern_ptr))
//              {
//                  pattern_ptr--; // Points to '*' again
//                  if (*filename_ptr) filename_ptr++; // Points to char after '.' f.e.
//              }
//              } else
//              return (FALSE); // FALSE only if no such
char exists: case Tufto.exe.zip '.'
//          }
//      }
//      else
//          return (FALSE); // Match failed */
//      }
//  }

```

// BELOW DISASM IS TO ENSURE ME FOR SAFETY OF (X && Y) WHERE Y WILL NOT BE EXECUTED UNLESS X IS TRUE.

```

//09 int main( int argc, char **argv ) {
//10     char
//11         *pattern_ptr = NULL, // Points to pattern */
//12         *filename_ptr = NULL; // Points to filename */
//13
//14         while (*filename_ptr && *filename_ptr != *pattern_ptr)
//15             filename_ptr++;
//16
//17     return( 0 );
//18 }

```

```

//_main PROC NEAR
//; File c:\program files\microsoft visual c++ toolkit 2003\yoshi.c
//; Line 12
//    xor     ecx, ecx
//; Line 14
//    mov     al, BYTE PTR [ecx]
//    push    esi
//    xor     esi, esi
//    test    al, al
//    je      SHORT $L562
//    mov     dl, BYTE PTR [esi]
//    npad     3
//$L549:
//    cmp     al, dl
//    je      SHORT $L562
//    mov     al, BYTE PTR [ecx+1]
//; Line 15
//    inc     ecx
//    test    al, al
//    jne     SHORT $L549
//$L562:
//; Line 17

```

```
//      xor    eax, eax
//      pop     esi
//; Line 18
//      ret     0
//_main    ENDP
```

```
char *Logo[] = {
"Yoshi(Filelist Creator), revision 06, written by Svalgyatchx,",
"in fact based on SWEEP.C from 'Open Watcom Project', thanks-thanks.",
"",
"Note1: So far, it works for current directory only.",
"Note2: Default method is depth-first traversal;",
"      may use pipe 'Yoshi|sort' for breadth-first_like traversal results.",
"Note3: Make notice that '*.*(extensionfull only)' is not equal to '*'(all);",
"      one disadvantage is an inability to list only extensionless filenames.",
"Note4: Search is case-insensitive as-must.",
"Note5: This revision allows multiple '*', and meaning of masks is:",
"      '?' - any character AND NOT EMPTY(default, for OR EMPTY see option -e);",
"      '*' - any character(s) or empty.",
"Note6: What is a .LBL(LineByLine) file?",
"      it is a bunch of GRAMMATICAL lines not mere LF or CRLF lines;",
"      it contains not symbols under 32(except CR and LF) and above 127;",
"      it contains not space symbol sequences.",
"Usage:",
"      Yoshi [option(s)] [filename(s)],
"      option(s):",
"      -v          i.e. verbose mode; output goes to console;",
"      -f          i.e. fullpath mode for output;",
"      -e          i.e. treat '?' as any character OR EMPTY;",
"      -t          i.e. touch all encountered files;",
"      -2          i.e. convert all encountered .TXT files to .LBL files;",
"      -o<filename> i.e. output goes to file(in append mode).",
"      filename(s):",
"      Wildcards '*' and wildcards '?' are allowed i.e. \"str*.c??\";",
"      default filename is '*'; DO NOT FORGET TO PUT",
"      filename(s) WITH WILDCARD(S) INTO QUOTE MARKS!",

// DAMN IGNORANCE LEADS TO(fix for below problem is QUOTE MARKS):
//"Note2: I am confused: 'Yoshi *.c' receives all(i.e.) *.c one-by-one in current",
//"      folder as command line parameters instead of WYSIWYG?! That leads",
//"      to lost wildcard(s) i.e. expanding them and nonsense-buggy results.",
//"      So, to avoid this ugly-pseudo-bug, the current folder must contain not",
//"      extensions you are looking(with wildcard(s)) for - GRMBL.",

"Examples:",
"      Yoshi -v -f -oCaterpillar_NON.lst \"*.lbl\" \"*.txt\" \"*.htm\" \"*.html\"",
"      Yoshi -f -oMyEbooks.txt \"*wiley*essential*.pdf\" \"*russian*.htm\"",
"",
      NULL
};

int verbose = 0;          /* verbose mode (default = false) */
int orempty = 0;
int mfullpath = 0;
int lbl = 0;
int tch = 0;
int moutput = 0;
unsigned long long int TotalSize = 0;
unsigned long int TotalFiles = 0, TotalFolders = 0;
unsigned long number_of_successful_touces = 0;
unsigned long int LBLFiles = 0;

struct utimbuf stamp;

//char    SaveDir[MAX_PATH];
```

```

char    SaveDir[MAX_DIR_PATH];

int Options_levels = INT_MAX;
unsigned Options_depthfirst = 1;

typedef struct dirstack {
    struct dirstack    *prev;
    char               name_len;
    char               name[_MAX_FNAME + _MAX_EXT];
}    dirstack;

dirstack    *Stack = NULL;
int         DoneFlag = 0;

```

```

void SetDoneFlag()
{
    DoneFlag = 1;
}

```

```

void *SafeMalloc( size_t n )
{
    void *p = malloc( n );
    if( p == NULL ) {
        puts( "Out of memory!" );
        SetDoneFlag();
    }
    return( p );
}

```

```

static char    CurrPathBuff[MAX_DIR_PATH + 2];

char *CurrPath()
{
    char        *p;
    dirstack    *stack;

    if( Stack->prev == NULL )
        return( "." );
    p = CurrPathBuff + MAX_DIR_PATH;
    *--p = '\\0';
    stack = Stack;
    for( ;; ) {
        p -= stack->name_len;
        memcpy( p, stack->name, stack->name_len );
        stack = stack->prev;
        if( stack->prev == NULL )
            break;
        *--p = '\\';
    }
    return( p );
}

```

```

char *StringCopy( char *dst, char *src )
{
    while( ( *dst = *src ) ) {
        ++dst;
        ++src;
    }
    return( dst );
}

```

```

//PUBLIC _StringCopy
//; Function compile flags: /Ogty
//_TEXT SEGMENT
//_dst$ = 8 ; size = 4
//_src$ = 12 ; size = 4
//_StringCopy PROC NEAR
//; File c:\program files\microsoft visual c++ toolkit 2003\sweep_me1_.c
//; Line 7
//      mov     edx, DWORD PTR _src$[esp-4]
//      mov     cl, BYTE PTR [edx]
//      test    cl, cl
//      mov     eax, DWORD PTR _dst$[esp-4]
//      mov     BYTE PTR [eax], cl
//      je      SHORT $L546
//$L545:
//      mov     cl, BYTE PTR [edx+1]
//; Line 8
//      inc     eax
//; Line 9
//      inc     edx
//      test    cl, cl
//      mov     BYTE PTR [eax], cl
//      jne     SHORT $L545
//$L546:
//; Line 12
//      ret     0
//_StringCopy ENDP

```

```

// TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL

```

```

void TXT2LBL(char *CurrentSolid)

```

```

{
    FILE *fp_unSOLID, *fp_SOLID;
    char LBLname[_MAX_FNAME + _MAX_EXT];
    char LBLname2[_MAX_FNAME + _MAX_EXT];
    long size_unSOLID, k;
    unsigned char workbyte, PREVworkbyte, LASTwrite_workbyte, c32 = ' ', c13 = '\r', c10 = '\n',
    cdot = '.';
    int NoLeadSPACE = 1;

```

```

StringCopy( LBLname, CurrentSolid );
LBLname[KAZE_strlen(LBLname)-1] = '1';
LBLname[KAZE_strlen(LBLname)-2] = 'B';
LBLname[KAZE_strlen(LBLname)-3] = 'L';
if( ( fp_unSOLID = fopen( CurrentSolid, "rb" ) ) == NULL )
{ printf( "Yoshi: Can't open %s file.\n", CurrentSolid ); exit( 1 ); }
if( ( fp_SOLID = fopen( LBLname, "wb" ) ) == NULL )
{ printf( "Yoshi: Can't open %s file.\n", LBLname ); exit( 1 ); }
fseek( fp_unSOLID, 0L, SEEK_END );
size_unSOLID = ftell( fp_unSOLID );
fseek( fp_unSOLID, 0L, SEEK_SET );
    workbyte = 0x00;
    LASTwrite_workbyte = 0x00;
    for( k = 0; k < size_unSOLID; k++ )
    {
        PREVworkbyte = workbyte;
        fread( &workbyte, 1, 1, fp_unSOLID );
        if( workbyte != c13 )
        {
            // a file can finish with no LF character!!!
            if (k != size_unSOLID - 1)
                if( workbyte != c10 )
                {
                    if (workbyte < c32)

```



```

        if (LASTwrite_workbyte != c32)
            fwrite( &c32, 1, 1, fp_SOLID );
            LASTwrite_workbyte = c32;
        }
    else if (workbyte >= 0x80)
    {
        if (LASTwrite_workbyte != c32)
            fwrite( &c32, 1, 1, fp_SOLID );
            LASTwrite_workbyte = c32;
        }
    else
    {
        if (LASTwrite_workbyte == c32 && workbyte == c32) {} else
            fwrite( &workbyte, 1, 1, fp_SOLID );
            LASTwrite_workbyte = workbyte;
        }
    }
    else
    {
        if (LASTwrite_workbyte != c32)
            fwrite( &c32, 1, 1, fp_SOLID );
            LASTwrite_workbyte = c32;
        }
    }
    else
    {
        if (LASTwrite_workbyte != '?' && LASTwrite_workbyte != '!' &&
LASTwrite_workbyte != '.') fwrite( &cdot, 1, 1, fp_SOLID );
        fwrite( &c13, 1, 1, fp_SOLID );
        fwrite( &c10, 1, 1, fp_SOLID );
    }
}
} // k 'for'
fclose(fp_SOLID);
fclose(fp_unSOLID);

```

```

StringCopy( LBLname2, LBLname );
LBLname2[KAZE_strlen(LBLname2)-1] = 'L';
LBLname2[KAZE_strlen(LBLname2)-2] = 'B';
LBLname2[KAZE_strlen(LBLname2)-3] = 'L';
if( ( fp_unSOLID = fopen( LBLname, "rb" ) ) == NULL )
{ printf( "Yoshi: Can't open %s file.\n", LBLname ); exit( 1 ); }
if( ( fp_SOLID = fopen( LBLname2, "wb" ) ) == NULL )
{ printf( "Yoshi: Can't open %s file.\n", LBLname2 ); exit( 1 ); }
fseek( fp_unSOLID, 0L, SEEK_END );
size_unSOLID = ftell( fp_unSOLID );
fseek( fp_unSOLID, 0L, SEEK_SET );
workbyte = 0x00;
for( k = 0; k < size_unSOLID - 2; k++ ) // -2 due to surely CRLF ending from LB1
{
    PREVworkbyte = workbyte;
    fread( &workbyte, 1, 1, fp_unSOLID );
    if(( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte ==
0x22)
    {
        fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
        fwrite( &workbyte, 1, 1, fp_SOLID );
        fwrite( &c13, 1, 1, fp_SOLID );
        fwrite( &c10, 1, 1, fp_SOLID );
        NoLeadSPACE = 1;
    }
    else if(( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte
== 0x27)
    {
        fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
        fwrite( &workbyte, 1, 1, fp_SOLID );
    }
}

```

```

        fwrite( &c13, 1, 1, fp_SOLID );
        fwrite( &c10, 1, 1, fp_SOLID );
        NoLeadSPACE = 1;
    }
    else if( ( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte
== 0x20)
    {
        fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
        fwrite( &c13, 1, 1, fp_SOLID );
        fwrite( &c10, 1, 1, fp_SOLID );
        NoLeadSPACE = 1;
    }
    else if( ( PREVworkbyte == '.' || PREVworkbyte == '?' || PREVworkbyte == '!' ) && workbyte
!= 0x20)
    {
        fwrite( &PREVworkbyte, 1, 1, fp_SOLID );
        fwrite( &c13, 1, 1, fp_SOLID );
        fwrite( &c10, 1, 1, fp_SOLID );
        NoLeadSPACE = 1;
        if( workbyte != '.' && workbyte != '?' && workbyte != '!' ) {
            fwrite( &workbyte, 1, 1, fp_SOLID );
            NoLeadSPACE = 0; }
        if (k == (size_unSOLID - 2) - 1)
        {
            fwrite( &workbyte, 1, 1, fp_SOLID ); // last char is .|?!
            fwrite( &c13, 1, 1, fp_SOLID );
            fwrite( &c10, 1, 1, fp_SOLID );
        }
    }
    else
    {
        if( workbyte != '.' && workbyte != '?' && workbyte != '!' ) {
            if( workbyte == c32 && NoLeadSPACE == 1 ) {} else {
                fwrite( &workbyte, 1, 1, fp_SOLID );
                NoLeadSPACE = 0; }}
        if (k == (size_unSOLID - 2) - 1)
        {
            fwrite( &workbyte, 1, 1, fp_SOLID ); // last char is .|?!
            fwrite( &c13, 1, 1, fp_SOLID );
            fwrite( &c10, 1, 1, fp_SOLID );
        }
    }
} // k 'for'
fclose(fp_SOLID);
fclose(fp_unSOLID);

if( remove( LBLname ) != 0 )
{ printf( "Yoshi: Can't remove %s file.\n", LBLname ); exit( 1 ); }
}
// TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL TXT2LBL

// Again thanks to TOUCH.C from WATCOM:
static int doTouchFile( char *full_name, struct utimbuf *stamp)
/*****/
{
    int utime_rc;

    utime_rc = utime( full_name, stamp );
    if( utime_rc == -1 ) {
    } else {
        return 1;
    }
    return 0;
}

```

```

void ProcessCurrentDirectory(char *WildCard, FILE *fp_outLOG)
{
    DIR                *dirh;
    struct dirent       *dp;
    dirstack            *stack;
    char cwd[MAX_DIR_PATH+1];    /* current working directory.      */

    if( !Options_depthfirst ) {
        //ExecuteCommands();
    }
    if( Options_levels != 0 ) {
        dirh = opendir( "." );
        if( dirh != NULL ) {
            --Options_levels;
            for( ;; ) {
                if( DoneFlag )
                    return;
                dp = readdir( dirh );
                if( dp == NULL )
                    break;

//#ifdef __UNIX__
                {
                    struct stat buf;
                    if ( stat( dp->d_name, &buf ) != 0 )
                    { // ##### CRASH SPOT ??? ?!!!! #####
                        // my case was for folders that are bad i.e. cannot be erase or copy
                        printf("Yoshi: Break after function 'stat'.\n");
                        exit( EXIT_FAILURE ); }
                    if( !S_ISDIR( buf.st_mode ) )
                    {
//if( S_ISREG( buf.st_mode ) && file_matches ( dp->d_name, WildCard ) ) // Under Windows it reads
//hidden and system files - grmbl.
                    if( S_ISREG( buf.st_mode ) && CompareWildCardWithName ( WildCard, dp->d_name, orempty ) )
                    {
                        TotalSize = TotalSize + buf.st_size;
                        TotalFiles++;

                        if (tch != 0) {
                            number_of_successful_touches +=
                                doTouchFile( dp->d_name, &stamp );
                        }

                        if (lbl != 0) {
                            KAZE_strrev(dp->d_name);
                            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] == '.') {
                                KAZE_strrev(dp->d_name);
                                LBLFiles++;
                                printf("Converting(LBLing) %s ...\n", dp->d_name);
                                TXT2LBL (dp->d_name);
                            } else
                                KAZE_strrev(dp->d_name);
                        }
                    }
                    if (mfullpath != 0) {
                        if ( !getcwd(cwd, MAX_DIR_PATH+1) ) {
                            // Some error
                        } else {
                            if (moutput != 0) {
                                fprintf( fp_outLOG, "%s\\%s\n", cwd, dp->d_name );
                            }
                            if (verbose != 0) printf("%s\\%s\n", cwd, dp->d_name);
                        }
                    }
                }
            }
        }
    }
    if (mfullpath != 0) {
        if ( !getcwd(cwd, MAX_DIR_PATH+1) ) {
            // Some error
        } else {
            if (moutput != 0) {
                fprintf( fp_outLOG, "%s\\%s\n", CurrPath(), dp->d_name );
            }
        }
    }
}

```

```

    }
    if (verbose != 0) printf("%s\\%s\n", CurrPath(), dp->d_name);
}
    continue;
}
}

//#else
//    if( !( dp->d_attr & _A_SUBDIR ) )
//        continue;
//endif

if( dp->d_name[0] == '.' ) {
    if( dp->d_name[1] == '.' || dp->d_name[1] == '\\0' )
        continue;
}
stack = SafeMalloc( sizeof( *stack ) );
if( DoneFlag )
    return;
stack->name_len = strlen( dp->d_name );
memcpy( stack->name, dp->d_name, stack->name_len + 1 );
stack->prev = Stack;
Stack = stack;

TotalFolders++;
//puts(CurrPath()); // All folders listing

    if (chdir( stack->name ) != 0)
    { // ##### CRASH SPOT ??? ?!!!! #####
        // my case was for hidden folder System Volume Information
        printf("Yoshi: Break after function 'chdir'.\n");
        exit( EXIT_FAILURE ); }
    ProcessCurrentDirectory( WildCard, fp_outLOG );
    chdir( ".." );
    Stack = stack->prev;
    free( stack );
}
++Options_levels;
closedir( dirh );
}
}
if( Options_depthfirst ) {
    //ExecuteCommands();
}
}

void PrintLogo()
{
    int i;

    for( i = 0; Logo[i] != NULL; ++i )
        puts( Logo[i] );
    //exit( EXIT_FAILURE );
}

//int scmp( unsigned char *s1, unsigned char *s2 )
//{
//    while( *s1 != '\\0' && *s1 == *s2 )
//    {
//        s1++;
//        s2++;
//    }
//    return( *s1-*s2 );
//}

```

```

void x64toaKAZE (          /* stdcall is faster and smaller... Might as well use it for the helper. */

```

```

unsigned long long val,
char *buf,
unsigned radix,
int is_neg
)
{
    char *p;           /* pointer to traverse string */
    char *firstdig;     /* pointer to first digit */
    char temp;          /* temp char */
    unsigned digval;     /* value of digit */

    p = buf;

    if ( is_neg )
    {
        *p++ = '-';      /* negative, so output '-' and negate */
        val = (unsigned long long) (- (long long) val);
    }

    firstdig = p;        /* save pointer to first digit */

    do {
        digval = (unsigned) (val % radix);
        val /= radix;     /* get next digit */

        /* convert to ascii and store */
        if (digval > 9)
            *p++ = (char) (digval - 10 + 'a'); /* a letter */
        else
            *p++ = (char) (digval + '0');      /* a digit */
    } while (val > 0);

    /* We now have the digit of the number in the buffer, but in reverse
       order. Thus we reverse them now. */

    *p-- = '\0';        /* terminate string; p points to last digit */

    do {
        temp = *p;
        *p = *firstdig;
        *firstdig = temp; /* swap *p and *firstdig */
        --p;
        ++firstdig;       /* advance to next two digits */
    } while (firstdig < p); /* repeat until halfway */
}

```

```

char * _ui64toaKAZEzerocomma (
    unsigned long long val,
    char *buf,
    int radix
)
{
    char *p;
    char temp;
    int txpman;
    int pxnman;
    x64toaKAZE(val, buf, radix, 0);
    p = buf;
    do {
    } while (*++p != '\0');
    p--; // p points to last digit
        // buf points to first digit
    buf[26] = 0;
    txpman = 1;
    pxnman = 0;
}

```

```

do
{ if (buf <= p)
{ temp = *p;
  buf[26-txpman] = temp; pxnman++;
  p--;
  if (pxnman % 3 == 0)
  { txpman++;
    buf[26-txpman] = (char) (',');
  }
}
else
{ buf[26-txpman] = (char) ('0'); pxnman++;
  if (pxnman % 3 == 0)
  { txpman++;
    buf[26-txpman] = (char) (',');
  }
}
txpman++;
} while (txpman <= 26);

return buf;
}

// Again thanks to TOUCH.C from WATCOM:
static void syncStamp( struct utimbuf *stamp )
/*****/
//struct tm
//{
//      int      tm_sec;      /* Seconds: 0-59 (K&R says 0-61?) */
//      int      tm_min;      /* Minutes: 0-59 */
//      int      tm_hour;      /* Hours since midnight: 0-23 */
//      int      tm_mday;      /* Day of the month: 1-31 */
//      int      tm_mon;      /* Months *since* january: 0-11 */
//      int      tm_year;      /* Years since 1900 */
//      int      tm_wday;      /* Days since Sunday (0-6) */
//      int      tm_yday;      /* Days since Jan. 1: 0-365 */
//      int      tm_isdst;     /* +1 Daylight Savings Time, 0 No DST,
//                             * -1 don't know */
//};
//_CRTIMP time_t __cdecl mktime (struct tm*);
//_CRTIMP struct tm* __cdecl localtime (const time_t*);

{
  time_t touch_time;
  static struct tm touchTime;

  time_t      curr_time;
  struct tm    *ptime;

  time( &curr_time );
  ptime = localtime( &curr_time );
  touchTime = *ptime;

  // touchTime.tm_sec = 0;
  // touchTime.tm_min = 0;
  // touchTime.tm_hour = 0;
  //touchTime.tm_mday = DateAdjust.day;
  //touchTime.tm_mon = DateAdjust.month - 1;
  //touchTime.tm_year = DateAdjust.year - 1900;
  //touchTime.tm_isdst = -1;

  touch_time = mktime( &touchTime );
  stamp->actime = touch_time;
  stamp->modtime = touch_time;
}

```

```

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

    char llToaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
    char *Wildcard = "*";
    char *out_file = "Yoshi.lst";
    FILE *fp_outLOG;

    PrintLogo();
    Stack = SafeMalloc( sizeof( *Stack ) );
    Stack->name_len = 1;
    Stack->prev = NULL;
    StringCopy( Stack->name, "." );
    getcwd( SaveDir, MAX_DIR_PATH+1 );
    //chdir( "d:\\www.sanmayce.com" );

    /*
    * loop for each option.
    * Stop if we run out of arguments
    * or we get an argument without a dash.
    */
    while ((argc > 1) && (argv[1][0] == '-')) {
        /*
        * argv[1][1] is the actual option character.
        */
        switch (argv[1][1]) {
            /*
            * -v verbose
            */
            case 'v':
                verbose = 1;
                break;
            case 'e':
                orempty = 1;
                break;
            case '2':
                lbl = 1;
                break;
            case 't':
                tch = 1;
                syncStamp( &stamp );
                break;
            case 'f':
                mfullpath = 1;
                break;
            /*
            * -o<name> output file
            * [0] is the dash
            * [1] is the "o"
            * [2] starts the name
            */
            case 'o':
                moutput = 1;
                out_file = &argv[1][2];
        }
        if( ( fp_outLOG = fopen( out_file, "ab" ) ) == NULL )
        { printf( "Yoshi: Can't open file %s.\n", out_file ); return( EXIT_FAILURE ); }

        break;
    }
    /*
    * -l<number> set max number of lines
    */
    //case 'l':
    //    line_max = atoi(&argv[1][2]);
    //    break;
    default:

```

```

        printf("Yoshi: Bad option %s\n", argv[1]);
        return( EXIT_FAILURE );
    }
    /*
     * move the argument list up one
     * move the count down one
     */
    ++argv;
    --argc;
}

/*
 * At this point all the options have been processed.
 * Check to see if we have no files in the list
 * and if so, we need to process just standard in.
 */
    if (argc == 1) {
ProcessCurrentDirectory( WildCard, fp_outLOG ); // *.* is default
    } else {
        while (argc > 1) {
            WildCard = argv[1];
ProcessCurrentDirectory( WildCard, fp_outLOG );
            ++argv;
            --argc;
        }
    }

if ((verbose == 1) && TotalFiles)
printf("\n");
else
if ((lbl == 1) && LBLFiles)
printf("\n");

printf("Yoshi: Total size of files: %s bytes.\n", _ui64toaKAZEzerocomma(TotalSize, llTOaDigits,
10)+(26-18)) ; // 26 are all 26-DESIRED=24
if (tch == 1)
printf("Yoshi: Total files: %s touched.\n", _ui64toaKAZEzerocomma(number_of_successful_touche,
llTOaDigits, 10)+(26-15)) ; // 26 are all 26-DESIRED=24
printf("Yoshi: Total files: %s.\n", _ui64toaKAZEzerocomma(TotalFiles, llTOaDigits, 10)+(26-15)) ;
// 26 are all 26-DESIRED=24
printf("Yoshi: Total folders: %s.\n", _ui64toaKAZEzerocomma(TotalFolders, llTOaDigits, 10)+(26-
13)) ; // 26 are all 26-DESIRED=24

    free( Stack );
    Stack = NULL;
    chdir( SaveDir );
if (moutput == 1) fclose( fp_outLOG );
    return( EXIT_SUCCESS );
}

```