

```

#include <stdio.h>
#include <strings.h>
#include <ctype.h>
#include <sys/param.h>

#define TABSIZE      8
#define COMMAND_CHAR  '.'
#define COL2         64
#define COL3         96

typedef char PATH[MAXPATHLEN];

FILE *master_index, *index_stb;

void ScanStoryboard ();
void ScanNameMap ();
char *Relativize ();
char *Absolutize ();

main (argc, argv)
    int argc;
    char *argv[];
    {
        char pathlist[4 * BUFSIZ];
        PATH path;
        int i;

        if (argc < 2)
            Error ("need pathname(s)");
        strcpy (pathlist, ". ");
        for (i = 2; i < argc; i++)
            {
                strcpy (path, argv[i]);
                if ((path[0] == '/') || (path[0] == '~'))
                    Absolutize (path, ".");
                else
                    Relativize (path, argv[1]);
                strcat (pathlist, path);
                strcat (pathlist, " ");
            }
        if (chdir (argv[1]))
            Error ("under-specified or bad first pathname");
        if (rename ("master-index", "master-index.old"))
            Error ("can't access master-index");
        if ((master_index = fopen ("master-index", "w")) == NULL)
            Error ("can't create master-index");
        unlink ("index.st0");
        if ((index_stb = fopen ("index.stx", "w")) == NULL)
            Error ("can't create index storyboard");
        fputs (".title\nMaster Index\n\n.synonyms\n!index\n\n.content\n\n",
            index_stb);
        fclose (index_stb);

        fputs ("----- ARTICLES ----- \n\n", master_index);
        fputs ("\"Master Index\"", master_index);
        TabTo (strlen ("\"Master Index\"", COL2, master_index);
        fputs (". /index.st0\n", master_index);
    }

```

```

fputs ("\\!index\\\"\\n", master_index);
ScanDirectories (ScanStoryboard, pathlist, "*.st[0-9]");

fputs ("----- PICTURES -----\\n\\n", master_index);
ScanDirectories (ScanNameMap, pathlist, "*.pn[0-9]");

fputs ("----- TARGETS -----\\n\\n", master_index);
ScanDirectories (ScanNameMap, pathlist, "*.tn[0-9]");

if (rename ("index.stx", "index.st0"))
    Error ("can't make index.st0");

exit (0);
}

```

```

ScanDirectories (ScanFunc, pathlist, pattern)
void (*ScanFunc) ();
char *pathlist, *pattern;
{
    FILE *filelist;
    char shell_command[4 * BUFSIZ], file[BUFSIZ];
    FILE *stream;
    int i;

    strcpy (shell_command, "find ");
    strcat (shell_command, pathlist);
    strcat (shell_command, "-name '");
    strcat (shell_command, pattern);
    strcat (shell_command, "' -print");

    if ((filelist = popen (shell_command, "r")) == NULL)
        Error ("can't run find");
    if ((index_stb = popen ("sort -f >> index.stx", "w")) == NULL)
        Error ("can't run sort");

    while (fgets (file, sizeof (file), filelist))
    {
        /* strip trailing newline */
        file[strlen (file) - 1] = '\\0';
        puts (file);
        if ((stream = fopen (file, "r")) == NULL)
            puts ("can't open");
        else
        {
            (*ScanFunc) (stream, file);
            fclose (stream);
        }
    }

    pclose (filelist);
    pclose (index_stb);
    fputs ("\\n\\n", master_index);

    return;
}

```

```

void ScanStoryboard (storyboard, file)
    FILE *storyboard;

```

```

char *file;
{
char arg[BUFSIZ];
long start, args, next;

start = 0;
FindCommand ("title ", &start, &args, &next, storyboard);
if ((start == EOF) || (args == next))
    printf ("missing title in %s\n", file);
else
    {
        GetArg (&args, arg, storyboard);
        fprintf (master_index, "\"%s\"", arg);
        if (arg[0] != '!')
            fprintf (index_stb, ".~ %s~ .nl\n", arg);
        TabTo (strlen (arg) + 2, COL2, master_index);
        fprintf (master_index, "%s\n", file);
        start = 0;
        FindCommand ("synonyms synonym ", &start, &args, &next,
            storyboard);
        while (args != next)
            {
                GetArg (&args, arg, storyboard);
                fprintf (master_index, "\"%s\"\n", arg);
            }
        putc ('\n', master_index);
    }

fclose (storyboard);

return;
}

```

```

void ScanNameMap (namemap, file)
FILE *namemap;
char *file;
{
char line[BUFSIZ];
long offset, length, args;
int c;

offset = 0;
while (offset != EOF)
    {
        args = offset;
        GetArg (&args, line, namemap);
        GetArg (&args, line, namemap);
        while (((c = getc (namemap)) != '\n') ||
            ((c = getc (namemap)) != '\n')) &&
            (c != EOF))
            ;
        if (c == EOF)
            length = -1;
        else
            length = ftell (namemap) - offset - 1;
        fprintf (master_index, "%s", line);
        TabTo (strlen (line), COL2, master_index);
        fprintf (master_index, "%s", file);
    }
}

```

```

        if ((offset > 0) || (length >= 0))
        {
            TabTo (COL2 + strlen (file), COL3, master_index);
            fprintf (master_index, "%d\t%d", offset, length);
        }
        fputs ("\n", master_index);
        if (c == EOF)
            offset = EOF;
        else
            offset += (length + 1);
    }

    fclose (namemap);

    return;
}

FindCommand (cmdlist, start, args, next, file)
char *cmdlist;
long *start, *args, *next;
FILE *file;
{
    char arg[BUFSIZ];
    int c;

    while (NextCommand (start, file), *start != EOF)
    {
        fscanf (file, "%*c%s", arg);
        if (InString (arg, cmdlist))
            break;
        *start = ftell (file);
    }
    while (c = getc (file), isspace (c))
        ;
    if (c == EOF)
        *next = *args = EOF;
    else
    {
        fseek (file, -1L, 1);
        *next = *args = ftell (file);
        NextCommand (next, file);
    }

    return;
}

NextCommand (pos, file)
long *pos;
FILE *file;
{
    int c;

    fseek (file, *pos, 0);
    while ((c = getc (file)) != EOF)
        if ((c == COMMAND_CHAR) &&
            ((c = getc (file)) != EOF) && !isspace (c))
        {
            fseek (file, -2L, 1);

```

```

        break;
    }
    if (c == EOF)
        *pos = EOF;
    else
        *pos = ftell (file);

    return;
}

int InString (s, list)
    char *s, *list;
{
    char *sp;
    int len;

    for (sp = s; *sp != '\0'; sp++)
        if (isupper (*sp))
            *sp = tolower (*sp);
    len = strlen (s);
    while (*list != '\0')
    {
        if (!strncmp (s, list, len) && (list[len] == ' '))
            break;
        list = index (list, ' ') + 1;
    }

    return (*list != '\0');
}

GetArg (args, arg, file)
    long *args;
    char *arg;
    FILE *file;
{
    int i;

    if (*args != EOF)
    {
        fseek (file, *args, 0);
        i = 0;
        while (((arg[i] = getc (file)) != '\n') && (arg[i] != EOF))
            ++i;
        /* strip trailing space */
        while ((--i >= 0) && isspace (arg[i]))
            ;
        arg[++i] = '\0';
        /* advance to next non-blank */
        while (((i = getc (file)) != EOF) && isspace (i))
            ;
        if (i == EOF)
            *args = EOF;
        else
        {
            fseek (file, -1L, 1);
            *args = ftell (file);
        }
    }
}

```

```

        else
            arg[0] = '\\0';

        return;
    }

TabTo (from, to, file)
    int from, to;
    FILE *file;
    {
        /* always at least one tab */
        putc ('\\t', file);
        from = from - (from % TABSIZE) + TABSIZE;
        for (; from < to; from += TABSIZE)
            putc ('\\t', file);

        return;
    }

Error (message)
    char *message;
    {
        fputs (message, stderr);
        fputs ("\\n", stderr);
        exit (1);
    }

char *Absolutize (path, base)
    char *path, *base;
    {
        PATH curr;

        getwd (curr);
        if (chdir (base))
            Error ("can't access path");
        if (chdir (path))
            Error ("can't access path");
        getwd (path);
        chdir (curr);
        return (path);
    }

char *Relativize (path, base)
    char *path, *base;
    {
        PATH abspath, absbase;
        char *p, *b;

        strcpy (abspath, path);
        strcpy (absbase, base);
        Absolutize (abspath, ".");
        Absolutize (absbase, ".");

        /* find common part */
        for (p = abspath, b = absbase; (*p == *b) && (*p != '\\0'); p++, b++)
            ;
        if (*p != '\\0')
            /* backup to directory marker */

```

```

        for (p--, b--; *p != '/'; p--, b--)
            ;
/* always start relative path with . */
strcpy (path, ".");
/* for every directory remaining in base, backup a level in path */
for (; *b != '\0'; b++)
    if (*b == '/')
        strcat (path, "/..");
/* now, can move down directory tree to end of path */
strcat (path, p);

return (path);
}

```