

\$SPAD/src/lib wct.c

The Axiom Team

July 28, 2014

Abstract

Contents

1	License	17
---	---------	----

— * —

```
/*
 * Word completion.
 *
 *
 * Word completion is driven from a list of completion tables. Each table
 * contains a list of words.
 *
 */

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

—————

The MACOSX platform is broken because no matter what you do it seems to include files from `[[/usr/include/sys]]` ahead of `[[/usr/include]]`. On linux systems these files include themselves which causes an infinite regression of includes that fails. GCC gracefully steps over that problem but the build fails anyway. On MACOSX the `[[/usr/include/sys]]` versions of files are badly broken with respect to the `[[/usr/include]]` versions.

— * —

```
#if defined(MACOSXplatform)
#include "/usr/include/unistd.h"
#else
#include <unistd.h>
#endif
#include <string.h>
#include <fcntl.h>
#if defined(MACOSXplatform)
#include "/usr/include/time.h"
#else
#include <time.h>
#endif
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>

/* #define PINFO */ /* A flag to suppress printing of the file info */

#define WCT /* A flag needed because ctype.h stole some
            * of my constants */

#include "edible.h"

#define MAX_PREFIX 1024
```

```

#define strneql(a,b,n)  (*(a) == *(b) && !strncmp((a),(b),(n)))
#define Delimiter(c) (! isalnum(c) && c != '%' && c != '!' && c != '?' && c != '_' )

#include "wct.h1"
#include "prt.h1"
#include "edin.h1"

static Wct *pwct = 0;
static Wix *pwix;
static Wix curr_wix;
static char curr_prefix[MAX_PREFIX];
static Wct *pHeadwct;

time_t
ftime(char *path)
{
    int rc;
    struct stat stbuf;

    rc = stat(path, &stbuf);
    if (rc == -1)
        fatal("Cannot determine status of %s.", path);

    return stbuf.st_mtime;
}

off_t
fsize(char *path)
{
    int rc;
    struct stat stbuf;

    rc = stat(path, &stbuf);
    if (rc == -1)
        fatal("Cannot determine status of %s.", path);

    return stbuf.st_size;
}

/*
 * Scan a word completion table for a particular word.
 */

Wix *
scanWct(Wct *pwct, char *prefix)

```

```

{
    long fmod;
    int preflen, i, wc;
    char **wv;

    preflen = strlen(prefix);
    strncpy(curr_prefix, prefix, MAX_PREFIX);

    pHeadwct = pwct;
    curr_wix.pwct = 0;
    curr_wix.word = 0;

    for (; pwct; pwct = pwct->next) {
        curr_wix.pwct = pwct;

        fmod = ftime(pwct->fname);
        if (fmod > pwct->ftime)
            reintern1Wct(pwct);

        wv = pwct->wordv;
        wc = pwct->wordc;
        for (i = 0; i < wc; i++) {
            curr_wix.word = i;
            if (strneql(wv[i], prefix, preflen))
                return &curr_wix;
        }
    }
    return 0;
}

Wix *
rescanWct(void)
{
    Wct *pwct, *start_pwct;
    int preflen, start_word, i, wc;
    char **wv, *prefix;

    start_pwct = curr_wix.pwct;
    start_word = curr_wix.word;

    if (!start_pwct) return(0);
    prefix = curr_prefix;
    preflen = strlen(prefix);

    /*
     * Finish the current structure.
     */

```

```

pwct = start_pwct;

curr_wix.pwct = pwct;
wv = pwct->wordv;
wc = pwct->wordc;
for (i = start_word + 1; i < wc; i++) {
    curr_wix.word = i;
    if (strneql(wv[i], prefix, preflen))
        return &curr_wix;
}

/*
 * Finish to the end of the list, doing whole structures.
 */
for (pwct = pwct->next; pwct; pwct = pwct->next) {
    curr_wix.pwct = pwct;

    wv = pwct->wordv;
    wc = pwct->wordc;
    for (i = 0; i < wc; i++) {
        curr_wix.word = i;
        if (strneql(wv[i], prefix, preflen))
            return &curr_wix;
    }
}

/*
 * Restart at beginning, doing whole structures.
 */
for (pwct = pHeadwct; pwct != start_pwct; pwct = pwct->next) {
    curr_wix.pwct = pwct;

    wv = pwct->wordv;
    wc = pwct->wordc;
    for (i = 0; i < wc; i++) {
        curr_wix.word = i;
        if (strneql(wv[i], prefix, preflen))
            return &curr_wix;
    }
}

/*
 * Do beginning half of the start structure.
 */
curr_wix.pwct = pwct;
wv = pwct->wordv;
wc = pwct->wordc;
for (i = 0; i <= start_word; i++) {
    curr_wix.word = i;
    if (strneql(wv[i], prefix, preflen))

```

```

        return &curr_wix;
    }

    /* Not found? */
    return 0;
}

/*
 * Summarize a table.
 */
void
skimWct(Wct *pwct)
{
    while (pwct) {
#ifdef PINFO
        skim1Wct(pwct);
#endif
        pwct = pwct->next;
    }
}

void
skim1Wct(Wct *pwct)
{
#define NHEAD    13
#define NTAIL    7

    int cc;

    printf("%-10s", pwct->fname);
    printTime((long *)&(pwct->ftime));
    cc = skimString(pwct->fimage, pwct->fsize, NHEAD, NTAIL);
    printf("%s", " " + (cc - (NHEAD + NTAIL)));
    printf(" [%d w, %ld c]", pwct->wordc, (long)(pwct->fsize));
    printf("\n");

#ifdef SHOW_WORDS
    {
        int i;
        char **wv;

        for (i = 1, wv = pwct->wordv; *wv; i++, wv++) {
            printf("%5d: %s\n", i, *wv);
        }
    }
#endif
}

void
printTime(long *clock)

```

```

{
    struct tm *tm;

    tm = localtime((time_t *)clock);
    printf("%.2d/%.2d/%.2d %.2d:%.2d:%.2d ",
           tm->tm_year, tm->tm_mon + 1, tm->tm_mday,
           tm->tm_hour, tm->tm_min, tm->tm_sec);
}

int
skimString(char *s, int slen, int nhead, int ntail)
{
    int spos, tlen, i, cc;

    cc = printf("\n");
    for (spos = 0; spos < slen && cc <= nhead; spos++)
        cc += prChar(s[spos]);

    /* Assume same tail has the same multi-character format ratio. */
    tlen = ntail * ((1.0 * spos) / nhead);

    if (spos + tlen >= slen)
        for (; spos < slen; spos++)
            cc += prChar(s[spos]);
    else {
        cc += printf("\n...\n");
        for (i = slen - tlen; i < slen; i++)
            cc += prChar(s[i]);
    }
    cc += printf("\n");
    return cc;
}

int
prChar(int c)
{
    if (c == '\n')
        return printf("\n");
    if (c == '\t')
        return printf("\t");
    if (c == '\b')
        return printf("\b");
    if (c == '"')
        return printf("\\");
    if (iscntrl(c))
        return printf("^%c", (c + 0100) % 0200);
    if (isascii(c))
        return printf("%c", c);

    return printf("\\%d", c);
}

```



```

}

Wct *
reread1Wct(Wct *pwct)
{
    int fd, rc;

    /* Check information about the file. */
    pwct->fsize = fsize(pwct->fname);
    pwct->ftime = ftime(pwct->fname);

    /* Allocate space for file image */
    if (pwct->fimage)
        free(pwct->fimage);
    pwct->fimage = (char *) malloc(pwct->fsize + 1);
    if (pwct->fimage == 0)
        sfatal("Cannot allocate new table.");
    pwct->fimage[pwct->fsize] = 0;

    /* Read file into buffer. */
    fd = open(pwct->fname, O_RDONLY);
    if (fd == -1)
        fatal("Cannot read file %s.", pwct->fname);
    rc = read(fd, pwct->fimage, pwct->fsize);
    if (rc != pwct->fsize)
        fatal("Did not read all of file %s.", pwct->fname);

    return pwct;
}

Wct *
read1Wct(char *fname)
{
    Wct *pwct;

    /* Allocate a new link for this file. */
    pwct = (Wct *) malloc(sizeof(Wct));
    if (pwct == 0)
        sfatal("Cannot allocate new table.");

    pwct->fname = fname;
    pwct->wordc = 0;
    pwct->wordv = 0;
    pwct->fimage = 0;
    pwct->next = 0;

    return reread1Wct(pwct);
}

Wct *

```

```

nconcWct(Wct *pwct, Wct * qwct)
{
    Wct *p0 = pwct;

    if (!p0)
        return qwct;

    while (pwct->next)
        pwct = pwct->next;
    pwct->next = qwct;

    return p0;
}

void
sortWct(Wct *pwct)
{
    while (pwct) {
        sort1Wct(pwct);
        pwct = pwct->next;
    }
}

void
sort1Wct(Wct *pwct)
{
    qsort((char *) pwct->wordv, pwct->wordc,
          sizeof(*(pwct->wordv)), mystrcmp);
}

int
mystrcmp(const void *s1, const void * s2)
{
    return strcmp(*(char **)s1, *(char **)s2);
}

/*
 * Break wct struct into words.
 */

void
burstWct(Wct *pwct)
{
    while (pwct) {
        burst1Wct(pwct);
        pwct = pwct->next;
    }
}

void

```

```

burst1Wct(Wct *pwct)
{
    char *s, **wv;
    int i, j, inword = 0;

    for (s = pwct->fimage, i = 0; i < pwct->fsize; s++, i++) {
        if (isspace(*s) || iscntrl(*s)) {
            *s = 0;
            inword = 0;
        }
        else if (!inword) {
            inword = 1;
            pwct->wordc++;
        }
    }

    if (pwct->wordv)
        free(pwct->wordv);
    pwct->wordv = (char **) calloc(pwct->wordc + 1, sizeof(char *));
    if (!pwct->wordv)
        fatal("Could not make word list for %s.", pwct->fname);

    s = pwct->fimage;
    i = 0;
    for (wv = pwct->wordv, j = 0; j < pwct->wordc; wv++, j++) {
        while (i < pwct->fsize && !s[i])
            i++;
        *wv = s + i;
        while (i < pwct->fsize && s[i])
            i++;
    }
    *wv = 0;
}

Wct *
intern1Wct(char *fname)
{
    Wct *pwct;

    pwct = read1Wct(fname);
    burst1Wct(pwct);
    return pwct;
}

void
reintern1Wct(Wct *pwct)
{
    reread1Wct(pwct);
    burst1Wct(pwct);
}

```

```

}

void
sfatal(char *s)
{
    fatal("%s", s);
}

void
fatal(char *fmt, char * s)
{
    static char fbuf[256];

    sprintf(fbuf, fmt, s);

    perror(fbuf);
    exit(1);
}

/* load up the wct database */
void
load_wct_file(char *fname)
{
    pwct = nconcWct(intern1Wct(fname), pwct);
}

void
skim_wct(void)
{
    skimWct(pwct);
}

/*
 * This routine is called when a tab is hit. It simply takes the current
 * buffer and tries to find a completion of the last word on the line in the
 * data base.
 */

void
rescan_wct(void)
{
    int b = curr_pntr - 1;
    int old_len;
    int new_len;
    int diff;
    int i;
    int ncs = 0;

```

```

/*
 * first thing I should do is find my way back to the beginning of the
 * word
 */
while (b && !Delimiter(buff[b]))
    b--;
if (Delimiter(buff[b]))
    b++;

old_len = curr_pntr - b;

pwix = rescanWct();

if (!pwix) {
    putchar(_BELL);
    fflush(stdout);
}
else {
    Wct *pwct = pwix->pwct; /* start replacing it */

    new_len = strlen(pwct->wordv[pwix->word]);
    if (new_len > old_len) {

        /*
         * I have to just slide the characters forward a bit, stuff in
         * the new characters, and then adjust curr_pntr
         */
        diff = new_len - old_len;
        if (curr_pntr != buff_pntr) {
            forwardcopy(&buff[curr_pntr + diff],
                        &buff[curr_pntr],
                        buff_pntr - curr_pntr);
            forwardflag_cpy(&buff_flag[curr_pntr + diff],
                           &buff_flag[curr_pntr],
                           buff_pntr - curr_pntr);
        }
        buff_pntr += diff;
        ncs = curr_pntr + diff;

        /* Now insert the new word */
        for (i = 0; i < new_len; i++)
            buff[b + i] = (pwct->wordv[pwix->word])[i];

        /* move cursor to the beginning of the word */
        for (; curr_pntr != b; curr_pntr--)
            putchar(_BKSPC);

        /** now print the characters on the rest of the line */
        printbuff(curr_pntr, buff_pntr - curr_pntr);
    }
}

```

```

        /* now move back the number of characters I want to */
        for (i = buff_ptr; i != ncs; i--)
            putchar(_BKSPC);

        fflush(stdout);

        curr_ptr = ncs;
    }
    else if (new_len < old_len) {
        /* this time I simply copy backwards and do the substituting */
        diff = old_len - new_len;
        strncpy(&buff[curr_ptr - diff],
                &buff[curr_ptr],
                buff_ptr - curr_ptr);
        flagncpy(&buff_flag[curr_ptr - diff],
                &buff_flag[curr_ptr],
                buff_ptr - curr_ptr);
        buff_ptr -= diff;
        ncs = curr_ptr - diff;

        /* Now insert the new word */
        for (i = 0; i < new_len; i++)
            buff[b + i] = (pwct->wordv[pwix->word])[i];

        /* move cursor to the beginning of the word */
        for (; curr_ptr != b; curr_ptr--)
            putchar(_BKSPC);

        /** now print the characters on the rest of the line */
        printf(b, buff_ptr - b);

        /* now blank out the characters out on the end of this line */
        for (i = 0; i < diff; i++)
            myputchar(' ');

        /* now move back the number of characters I want to */
        for (i = buff_ptr + diff; i != ncs; i--)
            putchar(_BKSPC);

        fflush(stdout);

        curr_ptr = ncs;
    }
    else {
        diff = 0;
        ncs = curr_ptr;
        /* Now insert the new word */
        for (i = 0; i < new_len; i++)
            buff[b + i] = (pwct->wordv[pwix->word])[i];
    }

```

```

        /* move cursor to the beginning of the word */
        for (; curr_pntr != b; curr_pntr--)
            putchar(_BKSPC);

        /** now print the characters on the rest of the line */
        printf(curr_pntr, buff_pntr - curr_pntr);

        /* now move back the number of characters I want to */
        for (i = buff_pntr; i != ncs; i--)
            putchar(_BKSPC);

        fflush(stdout);

        curr_pntr = ncs;
    }
}

void
find_wct(void)
{
    char search_buff[100];
    char *filler = search_buff;
    int b = curr_pntr - 1;
    int e = curr_pntr;
    int ne = 0;
    int st;
    Wix *pwix;
    int curr_len;
    int new_len;
    int diff;
    int i;

    /*
     * First thing I do is try and construct the string to be searched for.
     * Basically I just start from the curr_pntr and search backward until I
     * find a blank. Once I find a blank I copy forward until I get back to
     * curr_pntr;
     */
    if (!curr_pntr) {
        putchar(_BELL);
        return;
    }
    /* then get back to the first blank or back to the beginning */
    while (b && !Delimiter(buff[b]))
        b--;
    if (Delimiter(buff[b]))
        b++;

```

```

/* At the same time, let me find the end of the word */
while (e < buff_pntr && !Delimiter(buff[e])) {
    e++;
    ne++;
}

st = b;
curr_len = e - b;

/* now simply copy the text forward */
while (b < curr_pntr)
    *filler++ = buff[b++];

*filler = '\0';

pwix = scanWct(pwct, search_buff);

/*
 * else pwix = rescanWct();
 */

if (!pwix) {
    putchar(_BELL);
    fflush(stdout);
}
else {
    Wct *pwct = pwix->pwct;

    /*
     * printf("Found %s in file %s\n", pwct->wordv[pwix->word],
     * pwct->fname);
     */
    /* copy them buffer into where it should be */
    new_len = strlen(pwct->wordv[pwix->word]);
    diff = new_len - curr_len;
    if (curr_pntr != buff_pntr) {
        forwardcopy(&buff[curr_pntr + diff],
                    &buff[curr_pntr],
                    buff_pntr - curr_pntr);
        forwardflag_cpy(&buff_flag[curr_pntr + diff],
                        &buff_flag[curr_pntr],
                        buff_pntr - curr_pntr);
    }
    buff_pntr += diff;

    /* Now insert the new characters */
    for (i = new_len - diff; i < new_len; i++)
        buff[st + i] = (pwct->wordv[pwix->word])[i];
}

```



```

        /* Now move the cursor forward to the end of the word */
        for (i = 0; i < diff; i++)
            putchar(buff[curr_pntr++]);

        /** now print the characters on the rest of the line **/
        printbuff(curr_pntr, buff_pntr - curr_pntr);

        /* now move back the number of characters I want to */
        for (i = buff_pntr; i != e + diff; i--)
            putchar(_BKSPC);

        fflush(stdout);

        curr_pntr = diff + e;
    }

}

```

1 License

```

/*
Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd.
All rights reserved.

```

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical ALgorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER

OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*/

References

- [1] nothing