

\$SPAD/src/lib edin.c

The Axiom Team

July 28, 2014

**Abstract**

# Contents

1	License	3
---	---------	---

# 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.

```
*/
```

— \* —

```
/* #define debug 1 */
```

```
#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 <stdio.h>
#include <sys/types.h>

#include "edible.h"

#define HFT 0
#define SUN 1
#define DEC 2
#define control_to_alpha(x)  (x + ('A' - 0x01))
#define alpha_to_control(x)  (x - ('A' - 0x01))

int termId;
QueStruct *ring = NULL;
QueStruct *current = NULL;
int ring_size = 0;
int MAXRING = 64;
int prev_check = 10;
int curr_pntr;
int num_pntr;
int num_proc;
int had_tab;
int had_tab_last;
extern char buff[1024];          /* Buffers for collecting input and */
extern int buff_flag[1024]; /* flags for whether buff chars
        are printing or non-printing */
int buff_pntr;                  /* present length of buff */

#include "edin.h1"
#include "prt.h1"
#include "wct.h1"
#include "cursor.h1"
#include "fnct-key.h1"

void
init_reader(void)
{
    char *termVal;

```

```

buff[50] = '\0';          /** initialize some stuff ***/
init_flag(buff_flag, MAXLINE);
buff_pntr = curr_pntr = 0;

had_tab = 0;
had_tab_last = 0;
termVal = (char *) getenv("TERM");
if (!strcmp("sun", termVal))
    termId = SUN;
else if (!strcmp("xterm", termVal) || !strcmp("vt", termVal, 2))
    termId = DEC;
else if (!strcmp("hft", termVal) || !strcmp("aixterm", termVal, 7))
    termId = HFT;
}

void
do_reading(void)
{
    int ttt_read;
    int done_completely;

    done_completely = 0;
    num_proc = 0;
    while (num_proc < num_read) {
        if(in_buff[num_proc]== _ERASE) {
            back_over_current_char();
            num_proc++;
        }
        else {
            switch (in_buff[num_proc]) {
/* lets start checking for different types of chars */
                case _EOLN:
                case _CR:
/* If I have read a complete line, so send it to the child */
                send_line_to_child();
                if (!PTY)
                    myputchar('\n');
                break;

/*
 * Use 0x7f as delete
 */
                case _DEL:
/* Had a delete key */
                delete_current_char();
                break;

                case _CNTRL_W:
                move_back_word();

```

```

num_proc++;
break;
        case _TAB:
had_tab = 1;
/* command completion stuff */
num_proc++;
if (had_tab_last)
    rescan_wct();
else
    find_wct();
break;
        case _BELL:
insert_buff_nonprinting(1);
putchar(_BELL);
fflush(stdout);
break;
        case _ESC:

/*
 * get 2 characters more
 */
while (!(num_read - num_proc > 2)) {
    ttt_read = read(0,
in_buff + num_read,
2 - (num_read - num_proc) + 1);
    if (ttt_read > 0)
        num_read = num_read + ttt_read;
}
if ((in_buff[num_proc + 1] == _LBRACK)) {

/* ESC [ */

switch (in_buff[num_proc + 2]) {
    /* look for arrows */
case _A:
    /* up arrow */

/*
 * The first thing I plan to do is get rid of the present
 * input **
 */
prev_buff();
curr_pntr = buff_pntr;
num_proc = num_proc + 3;
break;
case _B:
    /* down arrow */
next_buff();
curr_pntr = buff_pntr;
num_proc = num_proc + 3;

```

```

        break;
case _C:
    /* right arrow */
    move_ahead();
    num_proc = num_proc + 3;
    break;
case _D:
    /* left arrow */
    move_back();
    num_proc = num_proc + 3;
    break;

/*
 * Use ^[[P as delete
 */
case _P:
    /* Had a delete key      ****/
    delete_current_char();
    break;
case _H:
case 0:
    move_home();
    num_proc += 3;
    break;
case _M:
case _Z:
    insert_buff_nonprinting(3);
    done_completely = 1;
    num_proc += 3;
    break;
case _x:
    num_proc = num_read;
    break;
case _1:
case _2:
case _0:

/*
 * I have had a possible function key hit, look for the
 * ones I want. check for ESC ] x ~
 */
while (!(num_read - num_proc > 3)) {
    ttt_read = read(0,
        in_buff + num_read,
        3 - (num_read - num_proc) + 1);
    if (ttt_read > 0)
num_read = num_read + ttt_read;
}
    if (in_buff[num_proc + 3] == _twiddle) {

```

```

        /*
         * treat ESC ] x ~
         */
        switch (in_buff[num_proc + 2]) {
        case _2:
flip(INS_MODE);
if (INS_MODE)
    Cursor_shape(5);
else
    Cursor_shape(2);
reprint(curr_pntr);
num_proc += 4;
break;
        default:
insert_buff_nonprinting(1);
break;
    }
    break;
}
/* check for ESC ] x y ~ */
while (!(num_read - num_proc > 4)) {
    ttt_read = read(0,
        in_buff + num_read,
        4 - (num_read - num_proc) + 1);
    if (ttt_read > 0)
num_read = num_read + ttt_read;
}
if (in_buff[num_proc + 4] == _twiddle) {

    /*
     * treat ESC ] x y ~
     */
    insert_buff_nonprinting(1);
    break;
}

/* check for ESC ] x y z [q|z] */

while (!(num_read - num_proc > 5)) {
    ttt_read = read(0,
        in_buff + num_read,
        5 - (num_read - num_proc) + 1);
    if (ttt_read > 0)
num_read = num_read + ttt_read;
}
if (insert_toggle(&in_buff[num_proc + 3])) {
    flip(INS_MODE);
    if (INS_MODE)
Cursor_shape(5);
    else

```



```

Cursor_shape(2);
    reprint(curr_pntr);
    num_proc = num_proc + 6;
    break;
}
else if (cntrl_end(&in_buff[num_proc + 3])) {
    num_proc = num_proc + 6;
    delete_to_end_of_line();
    break;
}
else if (back_word(&in_buff[num_proc + 3])) {
    move_back_word();
    num_proc += 6;
    break;
}
else if (fore_word(&in_buff[num_proc + 3])) {
    move_fore_word();
    num_proc += 6;
    break;
}
else if (end_key(&in_buff[num_proc + 3])) {
    move_end();
    num_proc += 6;
    break;
}
switch (in_buff[num_proc + 5]) {
case _q:

    /*
     * IBM function keys
     */
    {
char num[3];
int key;

num[0] = in_buff[num_proc + 3];
num[1] = in_buff[num_proc + 4];
num[2] = '\0';
key = atoi(num);
if (key > 0 && key < 13) {
    if (function_key[key].str != NULL) {
        handle_function_key(key, contNum);
        done_completely = 1;
    }
    else {
        insert_buff_nonprinting(6);
        done_completely = 1;
    }
}
}
else {

```

```

        insert_buff_nonprinting(6);
        done_completely = 1;
    }
    break;
    }
    case _z:

        /*
         * Sun function keys
         */
        {
            char num[3];
            int key;

            num[0] = in_buff[num_proc + 3];
            num[1] = in_buff[num_proc + 4];
            num[2] = '\0';
            key = atoi(num) - 23;
            if (key > 0 && key < 13) {
                if (function_key[key].str != NULL) {
                    handle_function_key(key, contNum);
                    done_completely = 1;
                }
                else {
                    insert_buff_nonprinting(6);
                    done_completely = 1;
                }
            }
            else if (atoi(num) == 14) {
                move_home();
                num_proc += 6;
                done_completely = 1;
            }
            else if (atoi(num) == 20) {
                move_end();
                num_proc += 6;
                done_completely = 1;
            }
            else if (atoi(num) == 47) {
                flip(INS_MODE);
                if (INS_MODE)
                    Cursor_shape(5);
                else
                    Cursor_shape(2);
                reprint(curr_ptr);
                num_proc = num_proc + 6;
                done_completely = 1;
            }
            else {
                insert_buff_nonprinting(6);

```

```

        done_completely = 1;
    }

    break;
}

    default:
        insert_buff_nonprinting(1);
        break;
}
default:
    if (!done_completely)
        insert_buff_nonprinting(1);
    break;
}
} /* if */
else { /* ESC w/o [ */
    insert_buff_nonprinting(1);
}
break;

    case _BKSPC:
back_over_current_char();
num_proc++;
break;
    default:
if (in_buff[num_proc] == _KILL) {
    delete_line();
    num_proc++;
}
else {
    if ((in_buff[num_proc] == _INTR) || (in_buff[num_proc] == _QUIT)) {
        write(contNum, &in_buff[num_proc], num_read - num_proc);
        if (!PTY)
            write(contNum, "\n", 1);
        num_proc++;
    }
    else {
        if (in_buff[num_proc] == _EOF) {
            insert_buff_nonprinting(1);
            if (!PTY)
write(contNum, "\n", 1);

            /*comment out this bit
if (!buff_ptr) {
write(contNum, &in_buff[num_proc], 1);
if (!PTY)
write(contNum, "\n", 1);
}
else {

```

```

write(contNum, buff, buff_pntr);
}
*/
    num_proc++;
}
else {
    if (in_buff[num_proc] == _EOL) {
send_line_to_child();
if (!PTY)
    write(contNum, "\n", 1);
    }
    else {
if (in_buff[num_proc] == _ERASE) {
    back_over_current_char();
    num_proc++;
}
else {
    if (control_char(in_buff[num_proc]))
        insert_buff_nonprinting(1);
    else
        insert_buff_printing(1);
}
    }
}
}
/* close the default case */
break;
}
/* switch */
} /*else*/
if (had_tab) {
    had_tab_last = 1;
    had_tab = 0;
}
else
    had_tab_last = 0;

}
/* while */
}

void
send_line_to_child(void )
{
    static char converted_buffer[MAXLINE];
    int converted_num;

    /* Takes care of sending a line to the child, and resetting the
       buffer for new input */
}

```

```

back_it_up(curr_pntr);

/* start by putting the line into the command line ring */
if (buff_pntr)
    insert_queue();

/* finish the line and send it to the child */
buff[buff_pntr] = in_buff[num_proc];
buff_flag[buff_pntr++] = 1;
buff[buff_pntr] = '\0';
buff_flag[buff_pntr] = -1;

/*
 * Instead of actually writing the Line, I have to substitute in the
 * actual characters recieved
 */
converted_num =
    convert_buffer(converted_buffer, buff, buff_flag, buff_pntr);
write(contNum, converted_buffer, converted_num);

/** reinitialize the buffer */
init_flag(buff_flag, buff_pntr);
init_buff(buff, buff_pntr);
/** reinitialize my buffer pointers */
buff_pntr = curr_pntr = 0;

/** reset the ring pointer */
current = NULL;
num_proc++;
return;
}

int
convert_buffer(char *target, char *source, int * source_flag, int num)
{
    int i, j;

    /*
     * Until I get something wierd, just keep copying
     */
    for (i = 0, j = 0; i < num; i++, j++) {
        switch (source[i]) {
            case _CARROT:
                if (source_flag[i] == 1) {
                    target[j] = source[i];
                }
                else {
                    if (source[i + 1] == _LBRACK) {
                        target[j] = _ESC;
                        i++;
                    }
                }
            }
        }
    }
}

```

```

}
else if (source[i + 1] >= 'A' && source[i + 1] <= 'Z') {
    target[j] = alpha_to_control(source[i + 1]);
    i++;
}
    }
    break;
    case '?':
    default:
        target[j] = source[i];
    }
}
return j;
}

void
insert_buff_printing(int amount)
{
    int count;

    /* This procedure takes the character at in_buff[num_proc] and adds
       it to the buffer. It first checks to see if we should be inserting
       or overwriting, and then does the appropriate thing */

    if ((buff_pntr + amount) > 1023) {
        putchar(_BELL);
        fflush(stdout);
        num_proc += amount;
    }
    else {

        if (INS_MODE) {

            forwardcopy(&buff[curr_pntr + amount],
&buff[curr_pntr],
buff_pntr - curr_pntr);
            forwardflag_cpy(&buff_flag[curr_pntr + amount],
&buff_flag[curr_pntr],
buff_pntr - curr_pntr);
            for (count = 0; count < amount; count++) {
buff[curr_pntr + count] = in_buff[num_proc + count];
buff_flag[curr_pntr + count] = 1;
            }
            ins_print(curr_pntr, amount);
            buff_pntr = buff_pntr + amount;
        }
        else {
            for (count = 0; count < amount; count++) {
if (buff_flag[curr_pntr + count] == 2) {

```

```

    myputchar(buff[curr_pntr + count]);
    curr_pntr += count + 1;
    delete_current_char();
    /** fix num_proc affected by delete **/
    num_proc -= 3;
    curr_pntr -= count + 1;
    myputchar(_BKSPC);
}
buff[curr_pntr + count] = in_buff[num_proc + count];
buff_flag[curr_pntr + count] = 1;
    }
    myputchar(in_buff[num_proc]);
    if (curr_pntr == buff_pntr)
buff_pntr++;
    }
    num_proc = num_proc + amount;
    curr_pntr = curr_pntr + amount;
    fflush(stdout);
}
return;

}

void
insert_buff_nonprinting(int amount)
{
    int count;

    /** This procedure takes the character at in_buff[num_proc] and adds
        it to the buffer. It first checks to see if we should be inserting
        or overwriting, and then does the appropriate thing */

    /** it takes care of the special case, when I have an esc character */

    if ((buff_pntr + amount) > 1023) {
        myputchar(_BELL);
        fflush(stdout);
        num_proc += amount;
    }
    else {
        if (INS_MODE) {
            forwardcopy(&buff[curr_pntr + amount + 1],
&buff[curr_pntr],
buff_pntr - curr_pntr);
            forwardflag_cpy(&buff_flag[curr_pntr + amount + 1],
&buff_flag[curr_pntr],
buff_pntr - curr_pntr);
            /** now insert the special character **/
            switch (in_buff[num_proc]) {
                case _ESC:

```

```

/** in this case I insert a '^[' into the string */
buff[curr_pntr] = _CARROT;
buff_flag[curr_pntr] = 2;
buff[curr_pntr + 1] = _LBRACK;
buff_flag[curr_pntr + 1] = 0;
break;
    default:
if (control_char(in_buff[num_proc])) {
    buff[curr_pntr] = _CARROT;
    buff_flag[curr_pntr] = 2;
    buff[curr_pntr + 1] = control_to_alpha(in_buff[num_proc]);
    buff_flag[curr_pntr + 1] = 0;
}
else {
    /** What do I have ? */
    buff[curr_pntr] = '?';
    buff_flag[curr_pntr] = 2;
    buff[curr_pntr + 1] = in_buff[num_proc];
    buff_flag[curr_pntr] = 0;
    break;
}
    }
    /** Now add the normal characters */
    for (count = 1; count < amount; count++) {
buff[curr_pntr + count + 1] = in_buff[num_proc + count];
buff_flag[curr_pntr + count + 1] = 1;
    }
    ins_print(curr_pntr, amount + 1);
    buff_pntr = buff_pntr + amount + 1;
    }
    else {
    /** I am in the overstrike mode */
    switch (in_buff[num_proc]) {
        case _ESC:
/** in this case I insert a '^[' into the string */
buff[curr_pntr] = _CARROT;
buff_flag[curr_pntr] = 2;
buff[curr_pntr + 1] = _LBRACK;
buff_flag[curr_pntr + 1] = 0;
break;
            default:
if (control_char(in_buff[num_proc])) {
    buff[curr_pntr] = _CARROT;
    buff_flag[curr_pntr] = 2;
    buff[curr_pntr + 1] = control_to_alpha(in_buff[num_proc]);
    buff_flag[curr_pntr + 1] = 0;
}
else {
    /** What do I have ? */
    buff[curr_pntr] = '?';

```



```

        buff_flag[curr_pntr] = 2;
        buff[curr_pntr + 1] = in_buff[num_proc];
        buff_flag[curr_pntr] = 0;
        break;
    }

    }
    for (count = 1; count < amount; count++) {
    if (buff_flag[curr_pntr + count] == 2) {
        curr_pntr += count + 1;
        delete_current_char();
        /** fix num. processed form delete **/
        num_proc -= 3;
        curr_pntr -= count + 1;
    }
    buff[curr_pntr + count + 1] = in_buff[num_proc + count];
    buff_flag[curr_pntr + count + 1] = 1;
    }
    /** now print the characters I have put in **/
    printbuff(curr_pntr, amount + 1);
    }
    num_proc = num_proc + amount;
    curr_pntr = curr_pntr + amount + 1;
    if (curr_pntr > buff_pntr)
        buff_pntr = curr_pntr;
    }
    return;

}

void
prev_buff(void)
{

    /**
     * If the current command ring is NULL, then I should NOT clear the
     * current line. Thus my business is already done
     */
    if (ring == NULL)
        return;
    clear_buff();
    init_buff(buff, buff_pntr);
    init_flag(buff_flag, buff_pntr);

    if (current == NULL) {
        if (ring == NULL)
            return;
        current = ring;
    }
    else
        current = current->prev;
}

```

```

strcpy(buff, current->buff);
flagcpy(buff_flag, current->flags);

/* first back up and blank the line */
fflush(stdout);
printbuff(0, strlen(buff));
curr_pntr = buff_pntr = strlen(buff);
fflush(stdout);
return ;
}

void
next_buff(void)
{
    /*
     * If the current command ring is NULL, then I should NOT clear the
     * current line. Thus my business is already done
     */
    if (ring == NULL)
        return;
    clear_buff();
    init_buff(buff, buff_pntr);
    init_flag(buff_flag, buff_pntr);
    if (current == NULL) {
        if (ring == NULL)
            return;
        current = ring->next;
    }
    else
        current = current->next;
    strcpy(buff, current->buff);
    flagcpy(buff_flag, current->flags);

    /* first back up and blank the line */
    fflush(stdout);
    printbuff(0, strlen(buff));
    curr_pntr = buff_pntr = strlen(buff);
    fflush(stdout);
    return ;
}

void
forwardcopy(char *buff1, char * buff2, int num)
{
    int count;

    for (count = num; count >= 0; count--)
        buff1[count] = buff2[count];
}

```

```

}

void
forwardflag_cpy(int *buff1,int * buff2,int  num)
{
    int count;

    for (count = num; count >= 0; count--)
        buff1[count] = buff2[count];
}

void
flagcpy(int *s,int *t)
{
    while (*t >= 0)
        *s++ = *t++;
    *s = *t;
}

void
flagncpy(int *s,int *t,int n)
{
    while (n-- > 0)
        *s++ = *t++;
}

void
insert_queue(void)
{
    QueStruct *trace;
    QueStruct *new;
    int c;

    if (!ECHOIT)
        return;
    if (ring != NULL && !strcmp(buff, ring->buff))
        return;
    for (c = 0, trace = ring; trace != NULL && c < (prev_check - 1);
        c++, trace = trace->prev) {
        if (!strcmp(buff, trace->buff)) {

            /*
             * throw this puppy at the end of the ring
             */
            trace->next->prev = trace->prev;
            trace->prev->next = trace->next;
            trace->prev = ring;
            trace->next = ring->next;
            ring->next = trace;

```

```

        trace->next->prev = trace;
        ring = trace;
        return;
    }
}

/*
 * simply places the buff command into the front of the queue
 */
if (ring_size < MAXRING) {
    new = (QueStruct *) malloc(sizeof(struct que_struct));
    if (new == NULL) {
        fprintf(stderr, "Malloc Error: Ran out of memory\n");
        exit(-1);
    }
    if (ring_size == 0) {
        ring = new;
        ring->prev = ring->next = new;
    }
    else {
        new->next = ring->next;
        new->prev = ring;
        ring->next = new;
        new->next->prev = new;
        ring = new;
    }
    ring_size++;
}
else
    ring = ring->next;

init_flag(ring->flags, MAXLINE);
init_buff(ring->buff, MAXLINE);
strcpy(ring->buff, buff);
flagncpy(ring->flags, buff_flag, buff_pntr);
(ring->buff)[buff_pntr] = '\0';
(ring->flags)[buff_pntr] = -1;
}

void
init_flag(int *flags, int num)
{
    int i;

    for (i = 0; i < num; i++)
        flags[i] = -1;
}

void
```

```

init_buff(char *flags, int num)
{
    int i;

    for (i = 0; i < num; i++)
        flags[i] = '\0';
}

void
send_function_to_child(void)
{
    /* Takes care of sending a line to the child, and resetting the
       buffer for new input */

    back_it_up(curr_pntr);
    /** start by putting the line into the command line ring */
    if (buff_pntr)
        insert_queue();

    /** finish the line and send it to the child */
    buff[buff_pntr] = _EOLN;

    buff_flag[buff_pntr++] = 1;
    buff[buff_pntr] = '\0';
    buff_flag[buff_pntr] = 0;
    write(contNum, buff, buff_pntr);

    /** reinitialize the buffer */
    init_flag(buff_flag, buff_pntr);
    init_buff(buff, buff_pntr);
    /** reinitialize my buffer pointers */
    buff_pntr = curr_pntr = 0;

    /** reset the ring pointer */
    current = NULL;

    num_proc++;
    return;
}

void
send_buff_to_child(int chann)
{
    if (buff_pntr > 0)
        write(chann, buff, buff_pntr);
    num_proc += 6;
    /** reinitialize the buffer */
    init_flag(buff_flag, buff_pntr);
    init_buff(buff, buff_pntr);
}

```

```
/** reinitialize my buffer pointers **/  
buff_pntr = curr_pntr = 0;  
/** reset the ring pointer **/  
current = NULL;  
return;  
}
```

\_\_\_\_\_

## References

- [1] nothing