$SPAD/src/lib prt.c

The Axiom Team

December 3, 2016

Abstract

Contents

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.
*/
```

—— * ——

```
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include "edible.h"

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

void
myputchar(char c)
{
```

```
    if (ECHOIT)
        putchar(c);
    return;
}

void
clear_buff(void)
{
    int count;

    /*** called when spadbuf gives me a line incase there is something already
      on the line ****/
    if (buff_pntr > 0) {
        /*** backup to the beginning of the line ***/
        for (count = curr_pntr; count > 0; count--)
            myputchar(_BKSPC);
        /** blank over the line      ***/
        for (count = 0; count < buff_pntr; count++) {
            myputchar(_BLANK);
        }
        /** back up again ***/
        for (count = buff_pntr; count > 0; count--)
            myputchar(_BKSPC);
        init_buff(buff, buff_pntr);
        init_flag(buff_flag, buff_pntr);
        curr_pntr = buff_pntr = 0;
    }
}


void
move_end(void)
{

    /** Moves cursor to the end of the line ***/
    if (curr_pntr == buff_pntr) {
        putchar(_BELL);
    }
    else {
        for (; curr_pntr < buff_pntr;) {
            myputchar(buff[curr_pntr++]);
        }
    }
    fflush(stdout);
}

void
move_home(void)
{
```

```
    /*** Moves the cursor to the front of the line ***/
    if (curr_pntr > 0) {
        for (; curr_pntr > 0;) {
            myputchar(_BKSPC);
            curr_pntr--;
        }
    }
    else {
        putchar(_BELL);
    }
    fflush(stdout);

}

void
move_fore_word(void)
{
    /** move the cursor to the next blank space  **/
    if (curr_pntr != buff_pntr) {
        myputchar(buff[curr_pntr]);
        curr_pntr++;
        while (curr_pntr < buff_pntr && buff[curr_pntr] != ' ') {
            myputchar(buff[curr_pntr]);
            curr_pntr++;
        }
    }
    else
        putchar(_BELL);
    fflush(stdout);
    return;
}

void
move_back_word(void)
{
    /*** moves the cursor to the last blank space ***/
    if (curr_pntr > 0) {
        myputchar(_BKSPC);
        curr_pntr--;
        while (curr_pntr > 0 && buff[curr_pntr - 1] != ' ') {
            myputchar(_BKSPC);
            curr_pntr--;
        }

    }
    else
        putchar(_BELL);
    fflush(stdout);
    return;
}
```

```
void
delete_current_char(void)
{
    /**  deletes the char currently above the current_pntr, if it can be **/
    if (curr_pntr != buff_pntr) {
        if (buff_flag[curr_pntr] == 1 || buff_flag[curr_pntr] == 0) {
            myputchar(_BLANK);
            myputchar(_BKSPC);
            strcpy(&buff[curr_pntr],
                   &buff[curr_pntr + 1]);
            flagcpy(&buff_flag[curr_pntr],
                    &buff_flag[curr_pntr + 1]);
            buff_pntr--;
            del_print(curr_pntr, 1);
        }
        else {
            /** lets delete two of the little buggers **/
            myputchar(_BLANK);
            myputchar(_BLANK);
            myputchar(_BKSPC);
            myputchar(_BKSPC);
            strcpy(&buff[curr_pntr],
                   &buff[curr_pntr + 2]);
            flagcpy(&buff_flag[curr_pntr],
                    &buff_flag[curr_pntr + 2]);
            buff_pntr -= 2;
            del_print(curr_pntr, 2);
        }
    }
    else {
        putchar(_BELL);
        fflush(stdout);
    }
    num_proc = num_proc + 3;
}

void
delete_to_end_of_line(void)
{
    int count;

    /*** deletes from the curr_pntr to the end of line ***/

    if (curr_pntr == buff_pntr)
        return;                 /** There is nothing to do **/

    /** blank over the end of the line      ***/
    for (count = curr_pntr; count < buff_pntr; count++) {
        myputchar(_BLANK);
```

```
    }
    /** back up again ***/
    for (count = buff_pntr; count > curr_pntr; count--)
        myputchar(_BKSPC);

    buff_pntr = curr_pntr;
    fflush(stdout);
    return;

}

void
delete_line(void)
{
    int count;

    /*** deletes the entire line                                  *****/

    if (buff_pntr == 0)
        return;                 /** There is nothing to do **/

    /** first I have to back up to the beginning of the line     ****/
    for (count = curr_pntr; count > 0; count--)
        myputchar(_BKSPC);

    /** blank over the end of the line      ***/
    for (count = 0; count < buff_pntr; count++) {
        myputchar(_BLANK);
    }
    /** back up again ***/
    for (count = buff_pntr; count > 0; count--)
        myputchar(_BKSPC);

    /* Also clear the buffer */
    init_buff(buff, buff_pntr);
    init_flag(buff_flag, buff_pntr);
    buff_pntr = curr_pntr = 0;

    fflush(stdout);
    return;

}

void
printbuff(int start,int  num)
{
    int trace;

    for (trace = start; trace < start + num; trace++)
        if (buff[trace] != '\0')
```

```
            myputchar(buff[trace]);
    fflush(stdout);
}

void
del_print(int start, int num)
{
    int count;

    /*** move the rest of the string     ***/
    for (count = start; count < buff_pntr; count++) {
        myputchar(buff[count]);
    }
    /** now blank out the number of chars we are supposed to ***/
    for (count = 0; count < num; count++)
        myputchar(_BLANK);
    /*** Now back up  ***/
    for (count = buff_pntr + num; count > start; count--)
        myputchar(_BKSPC);
    fflush(stdout);
}


void
ins_print(int start,int  num)
{
    int count;

    /** write the rest of the word ***/
    for (count = start; count < buff_pntr + num; count++) {
        myputchar(buff[count]);
    }
    /** now back up to where we should be ***/
    for (count = buff_pntr; count > start; count--)
        myputchar(_BKSPC);
    fflush(stdout);
}

void
reprint(int start)
{
    /**  simply reprints a single character **/
    if (buff[start] == '\0')
        myputchar(_BLANK);
    else
        myputchar(buff[start]);
    myputchar(_BKSPC);
    fflush(stdout);
    return;
}
```

```
void
back_up(int num_chars)
{
    int cnt;

    for (cnt = 0; cnt < num_chars; cnt++)
        myputchar(_BKSPC);
    for (cnt = 0; cnt < num_chars; cnt++)
        myputchar(_BLANK);
    for (cnt = 0; cnt < num_chars; cnt++)
        myputchar(_BKSPC);
    fflush(stdout);

}

void
back_it_up(int num_chars)
{
    int cnt;

    for (cnt = 0; cnt < num_chars; cnt++)
        myputchar(_BKSPC);
    fflush(stdout);
}


void
print_whole_buff(void)
{
    int trace;

    for (trace = 0; trace < buff_pntr; trace++)
        if (buff[trace] != '\0')
            myputchar(buff[trace]);
    fflush(stdout);
}

void
move_ahead(void)
{
    /*** simply moves the pointer ahead a single word ***/
    if (curr_pntr == buff_pntr) {
        putchar(_BELL);
    }
    else {
        if (buff_flag[curr_pntr] == 2) {
            myputchar(buff[curr_pntr++]);
        }
        myputchar(buff[curr_pntr++]);
```

```
    }
    fflush(stdout);
}

void
move_back(void)
{
    /** simply moves the cursor back one position **/
    if (curr_pntr == 0) {
        putchar(_BELL);
    }
    else {
        if (!buff_flag[curr_pntr - 1]) {
            myputchar(_BKSPC);
            curr_pntr--;
        }
        myputchar(_BKSPC);
        curr_pntr--;
    }
    fflush(stdout);
}

void
back_over_current_char(void)
{
    /*** simply backs over the character behind the cursor ***/
    if (curr_pntr == 0) {
        putchar(_BELL);
    }
    else {
        if (!buff_flag[curr_pntr - 1]) {
            myputchar(_BKSPC);
            myputchar(_BKSPC);
            myputchar(_BLANK);
            myputchar(_BLANK);
            myputchar(_BKSPC);
            myputchar(_BKSPC);
            strcpy(&buff[curr_pntr - 2],
                   &buff[curr_pntr]);
            flagcpy(&buff_flag[curr_pntr - 2],
                    &buff_flag[curr_pntr]);
            buff_pntr -= 2;
            curr_pntr -= 2;
            del_print(curr_pntr, 2);
        }
        else {
            myputchar(_BKSPC);
            myputchar(_BLANK);
            myputchar(_BKSPC);
            strcpy(&buff[curr_pntr - 1],
```

```
                &buff[curr_pntr]);
            flagcpy(&buff_flag[curr_pntr - 1],
                    &buff_flag[curr_pntr]);
            curr_pntr--;
            buff_pntr--;
            del_print(curr_pntr, 1);
        }
    }
    fflush(stdout);
    return;
}
```

References

[1] nothing