

## A source file de-tabifier for you

**Source:** <http://coding.derkeiler.com/Archive/C/ CPP/alt.comp.lang.learn.c-cpp/2004-05/0610.html>

---

**From:** Leor Zolman (*leor\_at\_bdsoft.com*)

**Date:** 05/16/04

Date: Sun, 16 May 2004 11:16:52 -0400

Dear allcc++ posters:

Lately I've been seeing a lot of posts coming in with long (>70 column) lines and messed-up tabs. This causes problems for folks trying to test the code, because long lines are wrapped indiscriminately by various stages of the posting mechanism, resulting in line breaks happening in inopportune places, and the "squashed" tabs make it very difficult to discern brace mismatch errors. The program I'm posting here will at least take care of the tabs being messed up, but the line length issue is one folks will have to deal with by conscious, pre-emptive source code formatting.

Following is `plist.c`, a program I originally wrote to help me format source listings for publication in CUJ articles while I worked on staff there. It takes your source file name as a command line parameter, along with options (if required), and outputs, on the standard output, a de-tabbed version of the program. If you redirect that into a file and copy the file literally into your postings, it will at least eliminate the tabbing issues (if caused by your news posting software. It can't fix code you've actually written that way; for that, you'd need some sort of "pretty printing" program. Or better yet, get into the habit of indenting your blocks!)

Note that you should configure `plist.c` to know what your default tabbing size is (4? 8?) by setting `TAB_SETTING`, so that's the default. You can still override that on the command line, if you wish. In fact, you can \*transform\* one tab setting to another; thus, you can write your code with tabs set to 8, and then process them with tabs set to 2, or 4, or whatever, to "squish" the code down for posting, but keeping the indentation consistent.

```
/******  
* plist.c: Process Source Listings for Publication  
* Written by Leor Zolman, 4/91, 11/91  
*  
* Usage:  
* plist [-t#[, @]] [-n[#]] [-c#] [-o] <file(s)>  
*  
* Options:  
* -t#[, @]: Set new tab spacing to every # columns
```

## alt.comp.lang.learn.c-c++: A source file de-tabifier for you

```
* If @ given, specifies OLD tab spacing
* (Both # and @ default to TAB_SETTING)
* -n[#]: Generate line numbers, using # as field
* width (defaults to NUM_WIDTH)
* -c#: Right-justify C-style comments to column #
* -o Write output into ".lst" file(s)
* If omitted, output goes to stdout.
* -r: Reset line numbers at each sequence of
* consecutive "/" in cols 1,2
*
* Compile:
* cl plist.c
*****/
```

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

#define NEEDSTR 0 /* 1 to use MY strstr() */
#define MAXLINE 200 /* longest allowable line */
#define TAB_SETTING 4 /* default soft tab setting */
#define NUM_WIDTH 3 /* default line # width */
#define TRUE 1
#define FALSE 0

char *makofname(char *, char *);
void do_line(char *);
void dofile(char *);

#if NEEDSTR
    char *strstr( char *, char * );
#endif

void pass1(char *, char *);
void pass2(char *);

int docmnts = FALSE; /* do comments */
int donums = FALSE; /* line numbering flag */
int make_lst = FALSE; /* create .LST files */
int reset_on_com = FALSE;
int saw_reset = FALSE;
int numwidth = NUM_WIDTH;
int tabstop = TAB_SETTING;
int old_tab = TAB_SETTING;
int cmnt_col; /* col. to right-justify to */

char fmt[20]; /* line numbering fmt spec */
FILE *fpi, *fpo; /* Input and output ptrs */
int lineno; /* current line number */
char cfname[50]; /* current file name */
```

```

int main(argc, argv)
int argc;
char **argv;
{
    int i, j;
                                /* Process command line options: */
    for (i = 1; i < argc; i++)
        if (argv[i][0] == '-')
            {
                switch (tolower(argv[i][1]))
                {
                    case 'n': donums = TRUE;
                            if (j = atoi(&argv[i][2]))
                                numwidth = j;
                            break;

                    case 't': j = sscanf(&argv[i][2], "%d,%d",
                                &tabstop, &old_tab);
                            break;

                    case 'c': docmnts = TRUE;
                            cmnt_col = atoi(&argv[i][2]);
                            if (cmnt_col < 30)
                                fprintf(stderr,
                                    "WARNING: -c value small!\n");
                            break;

                    case 'o': make_lst = TRUE;
                            break;

                    case 'r': reset_on_com = TRUE;
                            break;

                    default: fprintf(stderr, "Unknown option: '%s'\n",
                                argv[i]);
                            exit(0);
                }
            }
        for (j = i; j < argc - 1; j++) /* compress */
            argv[j] = argv[j + 1]; /* arg list */
        argc--;
        i--;
    }

    if (argc < 2)
        exit(fprintf(stderr,
            "Usage: pl [-t#[, @]] [-n[#]] [-c#] [-o] [-r] file(s)\n"));

    if (donums) /* create line number format string */
        sprintf(fmt, "%% %dd: ", numwidth);
}

```

```

for (i = 1; i < argc; i++)
    dofile(argv[i]); /* process list of given files */

return 0;
}

/*
 * Process the named file.
 * If make_lst is true, create a .LST file for output.
 * Otherwise, write output to standard output.
 */

void dofile(fname)
char *fname;
{
    char linbuf[MAXLINE];
    char ofname[30];

    strcpy(cfname, fname);
    if ((fpi = fopen(fname, "r")) == NULL)
    {
        fprintf(stderr,
            "\aCan't open input file \"%s\"\n", fname);
        return;
    }

    if (make_lst) /* if making .LST files... */
    {
        makofname(ofname, fname); /* create output filename */
        if ((fpo = fopen(ofname, "w")) == NULL)
        {
            fprintf(stderr, "\aCan't create \"%s\"\n", ofname);
            fclose(fpi);
            return;
        }
        fprintf(stderr, "Creating %s...\n", ofname);
    }
    else /* write to standard output */
        fpo = stdout;

    for (lineno=1; fgets(linbuf, MAXLINE, fpi); lineno++)
    {
        if (linbuf[strlen(linbuf) - 1] != '\n')
            exit(fprintf(stderr, "Line #%d too long.\n", lineno));
        do_line(linbuf); /* process each line */
    }

    fclose(fpi); /* close input file */
    if (make_lst) /* and, if writing .LST, */
        fclose(fpo); /* then close that too */
}

```

```

/*
 * do_line(): Process a line of text
 */

void do_line(cp)
char *cp;
{
    char result[6 * MAXLINE], *resultp;

    resultp = result;
    if (donums) /* if we're generating line nos. */
    {
        if (reset_on_com)
        {
            if (cp[0] == '/' && cp[1] == '/')
            {
                if (!saw_reset)
                {
                    lineno = 1;
                    fputs("\n", fpo);
                    saw_reset = TRUE;
                }
            }
            else
                saw_reset = FALSE;
        }
        sprintf(result, fmt, lineno);
        resultp += numwidth + 2;
    }

    pass1(cp, resultp); /* perform tab translation */
    if (docmnts && resultp != strstr(resultp, "/*"))
        pass2(result); /* right justify comments */

    fputs(result, fpo); /* write finished line */
}

/*
 * Perform tab translation for the line pointed to by cp.
 * Within comments, interpret soft tab as having spacing
 * of ORIGINAL soft tab setting, not the new tab setting.
 */

void pass1(cp, resultp)
char *cp, *resultp;
{
    char c;
    int col, old_col; /* current column no. */
    int in_cmnt = FALSE; /* if in a comment */

    col = old_col = 1;

```

```

while (c = *cp++)
{
    if (docmnts && !in_cmnt && c == '/' && *cp == '*')
        in_cmnt = TRUE;

    switch (c)
    {
        case '\t': /* process a tab: */
            if (!in_cmnt)
            { /* if not in a comment */
                do /* then add real spaces */
                    *resultp++ = ' ';
                while (col++ % tabstop);
                do /* and log virtual ones */
                    ;
                while (old_col++ % old_tab);
            }
            else /* if IN a comment, */
                do /* then translate as */
                    *resultp++ = ' '; /* per the OLD tab size */
                while (old_col++ % old_tab);
            break;

        default: /* handle other chars: */
            *resultp++ = c;
            col++; /* just pass on through */
            old_col++;
    }
}
*resultp = '\0';
}

/*
 * Justify comments to a common right margin.
 * Complain if there isn't enough room on a line to fit the
 * comment into the desired column width.
 */

void pass2(line)
char *line;
{
    char cmntbuf[2 * MAXLINE];
    char *cmnt_start, *cmnt_end;
    int cmnt_len, code_len, col;

    if (!(cmnt_start = strstr(line, "/*")) ||
        !(cmnt_end = strstr(line, "*/")))
    {
        if (strlen(line) > cmnt_col)
            fprintf(stderr,
                "\aWarning: %s: line %d too long.\n", cfname, lineno);
    }
}

```

## alt.comp.lang.learn.c-c++: A source file de-tabifier for you

```
    return; /* If no complete comment, return */
}

*(cmnt_end += 2) = '\0'; /* found a comment */
strcpy(cmntbuf, cmnt_start); /* save the comment */
cmnt_len = strlen(cmntbuf); /* get its length */

while (isspace(*--cmnt_start)) /* find last non- */
    ; /* space before the comment */
*++cmnt_start = '\0'; /* and terminate the code */

code_len = cmnt_start - line; /* length of code */
if (code_len + cmnt_len > cmnt_col) /* comment fit? */
{ /* no. complain. */
    fprintf(stderr,
        "\aWarning: %s: comment on line %d doesn't fit.\n",
        cfname, lineno);
    strcat(line, cmntbuf); /* and concatenate it */
}
else
{ /* comment will fit. Pad with spaces: */
    for (col = code_len;
        col < (cmnt_col - cmnt_len); col++)
        line[col] = ' ';
    strcpy(&line[col], cmntbuf); /* add comment */
}
strcat(line, "\n");
}

/*
 * Create output file name by replacing extension of
 * original filename with ".lst":
 */

char *makofname(dest, orig)
char *dest, *orig;
{
    int i;

    strcpy(dest, orig);
    for (i = 0; dest[i] && dest[i] != '.'; i++)
        ;
    strcpy(&dest[i], ".lst");
    return dest;
}

#if NEEDSTR
/*
 * Search for first occurrence of substring s2 in s1:
 * Return a pointer to the first occurrence, or NULL if
 * none was found.
 */
```

## alt.comp.lang.learn.c-c++: A source file de-tabifier for you

```
* (provided for Xenix only; this function is included
* with most ANSI-C distribution libraries)
*/

char *strstr(s1, s2)
char *s1, *s2;
{
    int i, j, nposs;
    char *p1;

    int len1 = strlen(s1);
    int len2 = strlen(s2);

    if (len1 < len2) /* can't have substring longer */
        return NULL; /* than entire string! */

    nposs = len1 - len2; /* # of possible positions */

    for (i = 0; i <= nposs; i++)
    { /* check each possible position */
        for (j = 0, p1 = &s1[i]; j < len2; j++)
            if (*p1++ != s2[j])
                break; /* break on 1st mismatch */
        if (j == len2) /* if no mismatches, */
            return &s1[i]; /* then pattern was found */
    }
    return NULL; /* didn't find the pattern */
}
#endif

--
Leor Zolman --- BD Software --- www.bdsoft.com
On-Site Training in C/C++, Java, Perl and Unix
C++ users: download BD Software's free STL Error Message Decryptor at:
www.bdsoft.com/tools/stlfilt.html
```