```
================================================================================
            /local/submit/submit/comp10002/ass1/hjthorpe/src/ass1test3.c
================================================================================

/* Solution to comp10002 Assignment 1, 2019 semester 2.

   Authorship Declaration:

   (1) I certify that the program contained in this submission is completely
   my own individual work, except where explicitly noted by comments that
   provide details otherwise.  I understand that work that has been developed
   by another student, or by me in collaboration with other students,
   or by non-students as a result of request, solicitation, or payment,
   may not be submitted for assessment in this subject.  I understand that
   submitting for assessment work developed by or in collaboration with
   other students or non-students constitutes Academic Misconduct, and
   may be penalized by mark deductions, or by other penalties determined
   via the University of Melbourne Academic Honesty Policy, as described
   at https://academicintegrity.unimelb.edu.au.

   (2) I also certify that I have not provided a copy of this work in either
   softcopy or hardcopy or any other form to any other student, and nor will
   I do so until after the marks are released. I understand that providing
   my work to other students, regardless of my intention or any undertakings
   made to me by that other student, is also Academic Misconduct.

   (3) I further understand that providing a copy of the assignment
   specification to any form of code authoring or assignment tutoring
   service, or drawing the attention of others to such services and code
   that may have been made available via such a service, may be regarded
   as Student General Misconduct (interfering with the teaching activities
   of the University and/or inciting others to commit Academic Misconduct).
   I understand that an allegation of Student General Misconduct may arise
   regardless of whether or not I personally make use of such solutions
   or sought benefit from such actions.

   Signed by: Hunter Thorpe 1079893
   Dated:     [Enter the date that you "signed" the declaration]

*/

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

#define LINE_LENG 50
#define LEFT_MARG 4
#define BLANK_START 1
#define NON_BLANK 0
#define MAX_LENG 1000

#define CH_BLANK ' '
#define CH_PERIOD '.'

#define NO 0
#define YES 1

/*******************************************************************************/
/* function prototypes */

int mygetchar();
char read_line(char *line);
void print_margin(int margin_spc);
int inspectchar(char character);
int char_to_int(char char_1, char char_2, char char_3);
void nullify_line(char *line);
void proccess_line(char *line_array, int line_len_var, int margin_var,
    int *char_counter, int *carry_over, int *margin_needed,
    char *recent_break, char *recent_para);
void break_command(int margin_var);
void para_command(int recent_break, int margin_var);
void left_marg_command(char *line_array, int *margin_var, int recent_para,
        int recent_break);
```

```
75  void line_len_command(char *line_array, int *line_len_var, int margin_var,
            int recent_break, int recent_para);
    void centre_line_command(char *line_array, int line_len_var, int *margin_needed,
            int margin_var, int recent_break, int recent_para, int carry_over,
            int char_counter);
80  void header_line_command(int *h_count_1, int *h_count_2, int *h_count_3,
            char *line_array, int margin_var, int recent_break, int recent_para,
            int line_len_var);
    /*****************************************************************************/

85  int
    main(int argc, char *argv[]) {

        int carry_over = 0;
        int char_counter = 0;
90      int margin_var = LEFT_MARG;
        int line_len_var = LINE_LENG;
        int h_count_1 = 0;
        int h_count_2 = 0;
        int h_count_3 = 0;
95      int plwh_update;
        int margin_needed = YES;
        char recent_para = YES;
        char recent_break = YES;
        char line_array[MAX_LENG];
100
        nullify_line(line_array);

        /* iterating through lines of stdin */
        while((read_line(line_array)) == YES) {
105         /* handiling lines that dont start with . */
            if (line_array[1] != CH_PERIOD) {
                proccess_line(line_array, line_len_var, margin_var, &char_counter,
                    &carry_over, &margin_needed, &recent_break, &recent_para);
            }
110

        /* handiling lines that start with . */
            else {
                plwh_update = NO; /*plwh stands for paragraph, left margin, width
115             * and header commands as they require the same status updates */

                /* break commands */
                if (line_array[2] == 'b' && recent_para == NO &&
                    recent_break == NO) {
120                 break_command(margin_var);
                    carry_over = 0;
                    char_counter = 0;
                    margin_needed = NO;
                    recent_break = YES;
125             }

                /* paragragh commands */
                if (line_array[2] == 'p' && recent_para == NO) {
                    para_command(recent_break, margin_var);
130                 plwh_update = YES;
                }

                /* left margin commands */
                if (line_array[2] == 'l') {
135                 left_marg_command(line_array, &margin_var, recent_para,
                        recent_break);
                    plwh_update = YES;
                }

140             /* width/line length commands */
                if (line_array[2] == 'w') {
                    line_len_command(line_array, &line_len_var, margin_var,
                        recent_break, recent_para);
                    plwh_update = YES;
145             }

                /* centre text commands */
                if (line_array[2] == 'c') {
```

```
                    centre_line_command(line_array, line_len_var, &margin_needed,
                        margin_var, recent_break, recent_para, carry_over,
                        char_counter);
                    margin_needed = YES;
                    recent_break = YES;
                    recent_para = NO;
                    char_counter = 0;
                    carry_over = 0;
                }

                /* header commands */
                if (line_array[2] == 'h') {
                    plwh_update = YES;
                    header_line_command(&h_count_1, &h_count_2, &h_count_3,
                        line_array, margin_var, recent_break, recent_para,
                        line_len_var);
                }
                if (plwh_update == YES) {
                    recent_para = YES;
                    recent_break = YES;
                    carry_over = 0;
                    char_counter = 0;
                    margin_needed = NO;
                }
            }
        /* counting character spaces used in each iteration to determine when to
        * place a new line */
        carry_over = char_counter + carry_over;
        char_counter = 0;
        }
        return 0;
}

/*****************************************************************************/

void
    ccess_line(char *line_array, int line_len_var, int margin_var,
        int *char_counter, int *carry_over, int *margin_needed,
        char *recent_break, char *recent_para) {

        int word_len = 0;
        int word_iter;
        int array_iter;
        int starting_char;

        if (*margin_needed == YES && line_array[1] != '\0') {
            print_margin(margin_var);
            *margin_needed = NO;
        }
        starting_char = NON_BLANK; /* 0 */
        if (line_array[1] == CH_BLANK) {
            starting_char = BLANK_START; /* 1 */
        }

        /* iterating through chars of line_array */
        for (array_iter = starting_char; line_array[array_iter] != '\0';
            array_iter++) {
            word_len = 0;
            if ((line_array[array_iter] == CH_BLANK) ||
                (starting_char == BLANK_START)) {

                /* counting length of word */
                for (word_iter = 1; (inspectchar(line_array[array_iter +
                    word_iter])) == 1; word_iter++) {
                    word_len = word_len + 1;
                }

                /* addding a new line if needed */
                if (word_len >= (line_len_var - *char_counter - *carry_over)) {
                    printf("\n");
                    *margin_needed = NO;
                    *recent_break = YES;
                    *char_counter = 0;
                    *carry_over = 0;
```

```
                        /* words longer than allowed length width */
                        if (word_len > line_len_var) {
                            print_margin(margin_var);
                            for (word_iter = 1; inspectchar(line_array[array_iter +
                                word_iter]) == 1; word_iter++) {
                                printf("%c", line_array[array_iter + word_iter]);
                            }
                            *recent_para = NO;
                            *recent_break = YES;
                            printf("\n");
                            *margin_needed = NO;
                            nullify_line(line_array);

                            /* if its the last word of the line */
                            if (line_array[array_iter + word_iter + 1] == '\0') {
                                *char_counter = *char_counter - 1;
                            }

                            array_iter = array_iter + word_iter ;
                        }
                        if (starting_char == NON_BLANK) {
                            array_iter = array_iter + 1;
                        }
                        print_margin(margin_var);
                        *margin_needed = NO;
                    }
                }

            /* to avoid printing a blank char as the first char of a line */
            if ((*carry_over + *char_counter == 0) && (line_array[array_iter] ==
                CH_BLANK)) {

                } else {
                if (isprint(line_array[array_iter])){

                    printf("%c", line_array[array_iter]);
                }
                *recent_para = NO;
                *recent_break = NO;
                *char_counter = *char_counter + 1;
                }
            }
        }

    /*****************************************************************************/

    void
        break_command(int margin_var) {

            printf("\n");
            print_margin(margin_var);
        }

    /*****************************************************************************/

    void
        para_command(int recent_break, int margin_var) {

            printf("\n");
            if (recent_break == NO) {
                printf("\n");
            }
            print_margin(margin_var);
        }

    /*****************************************************************************/

    void
        left_marg_command(char *line_array, int *margin_var, int recent_para,
            int recent_break) {
            int old_marg;

            old_marg = *margin_var;
```

```
            *margin_var = char_to_int(line_array[4], line_array[5],
                line_array[6]);
            if (recent_para == NO) {
                if (recent_break == NO) {
                    printf("\n");
                }
                printf("\n");
                print_margin(*margin_var);
            } else {
                print_margin(*margin_var - old_marg);
            }
        }

/*****************************************************************************/

void
    line_len_command(char *line_array, int *line_len_var, int margin_var,
        int recent_break, int recent_para) {

        *line_len_var = char_to_int(line_array[4], line_array[5],
            line_array[6]);
        if (recent_para == NO) {
            if (recent_break == NO) {
                printf("\n");
            }
            printf("\n");
            print_margin(margin_var);
        }
    }

/*****************************************************************************/

void
    centre_line_command(char *line_array, int line_len_var, int *margin_needed,
        int margin_var, int recent_break, int recent_para, int carry_over,
        int char_counter) {

        int c_line_counter = 0;
        int command_iter;
        int first_spaces;

        if (recent_break == NO) {
            printf("\n");
        }
        if (*margin_needed == YES || carry_over + char_counter != 0) {
            print_margin(margin_var);
            *margin_needed = NO;
        }
        for (command_iter = 4; line_array[command_iter] != '\0';
            command_iter++) {
            c_line_counter = c_line_counter + 1;
        }
        if (c_line_counter >= line_len_var - margin_var) {
            for (command_iter = 4; line_array[command_iter] != '\0';
                command_iter++) {
                printf("%c", line_array[command_iter]);
            }
        } else {
            first_spaces = ((line_len_var - c_line_counter) / 2);

            if (line_array[4] == CH_BLANK) {
                print_margin(margin_var);
            }
            for (command_iter = 0; command_iter < first_spaces;
                command_iter++) {
                printf("%c", CH_BLANK);
            }
            for (command_iter = 4; line_array[command_iter] != '\0';
                command_iter++) {
                printf("%c", line_array[command_iter]);
            }
        }
        printf("\n");
    }
```

```
/*****************************************************************************/

void
    header_line_command(int *h_count_1, int *h_count_2, int *h_count_3,
        char *line_array, int margin_var, int recent_break, int recent_para,
        int line_len_var) {

        int command_iter;
        int boundary_counter;

        if (recent_para == NO) {
            if (recent_break == NO) {
                printf("\n");
            }
            printf("\n");
            print_margin(margin_var);
        }

        if (line_array[4] == '1') {

            /* printing line of -'s */
            for (boundary_counter = 0; boundary_counter < line_len_var;
                boundary_counter++) {
                printf("%c", '-');
            }
            printf("\n");
            print_margin(margin_var);
            *h_count_1 = *h_count_1 + 1;
            *h_count_2 = 0;
            *h_count_3 = 0;
            printf("%c", *h_count_1 + '0');
        }
        if (line_array[4] == '2') {
            *h_count_2 = *h_count_2 + 1;
            *h_count_3 = 0;
            printf("%c", *h_count_1 + '0');
            printf("%c", CH_PERIOD);
            printf("%c", *h_count_2 + '0');
        }
        if (line_array[4] == '3') {
            *h_count_3 = *h_count_3 + 1;
            printf("%c", *h_count_1 + '0');
            printf("%c", CH_PERIOD);
            printf("%c", *h_count_2 + '0');
            printf("%c", CH_PERIOD);
            printf("%c", *h_count_3 + '0');
        }
        for (command_iter = 5; line_array[command_iter] != '\0';
            command_iter++) {
            printf("%c", line_array[command_iter]);
        }
        printf("\n");
        printf("\n");
        print_margin(margin_var);

    }

/*****************************************************************************/
/* mygetchar function obtained from assignment FAQ page */

int
    mygetchar() {
        int c;
        while ((c=getchar())=='\r') {
        }
        return c;
    }

void
    nullify_line(char *line) {
        int k;

        for (k=0; k < MAX_LENG; k++) {
```

```
445                 line[k] = '\0';
                }
        }

/*************************************************************************/
/* converts characters to ints for the purpose of margin and width commands */

int
    char_to_int(char char_1, char char_2, char char_3) {
        int int_1;
        int int_2;
        int int_3;

        if (isdigit(char_1) && isdigit(char_2) && isdigit(char_3)) {
            int_1 = char_1 - '0';
            int_2 = char_2 - '0';
            int_3 = char_3 - '0';
            return (int_1 * 100) + (int_2 * 10) + (int_3);
        }
        if (isdigit(char_1) && isdigit(char_2)) {
            int_1 = char_1 - '0';
            int_2 = char_2 - '0';
            return (int_1 * 10) + int_2;
        } else {
            int_1 = char_1 - '0';
            return int_1;
        }
    }

/*************************************************************************/
/* returns 0 if char is a space, null char or EOF, if not returns 1 */

int
    inspectchar(char character) {
        int char_status = 0;

        if (character != CH_BLANK && character != EOF && character != '\0') {
            char_status = 1;
        }
        return char_status;
    }

/*************************************************************************/

/* prints out margin based off given integer */

void
    print_margin(int margin_spc) {
        int j;
        for (j=0; j < margin_spc; j++) {
                printf("%c", CH_BLANK);
            }
    }

/*************************************************************************/
/* reads line from stdin returns NO when it reaches EOF, YES otherwise */

char
    read_line(char *line) {
        int line_iter = 1;
        char run_again = YES;
        char pot_char;
        char prev_char = 'a'; /* initiliasing to non space char */

        line[0] = CH_BLANK;
        while ((pot_char = mygetchar()) != '\n') {
            if (pot_char == EOF) {
                run_again = NO;
                break;
            }

            /* turning all white space into standard space char */
            if ((pot_char == '	') || (pot_char == '    ') ||
                    (pot_char == ' ') || (pot_char == '  ')) {
```

```
                        pot_char = CH_BLANK;
                }

                /* preventing duplicate spaces */
                if ((prev_char == CH_BLANK) && (pot_char == CH_BLANK)) {
                } else {
                line[line_iter] = pot_char;
                line_iter = line_iter + 1;
                }
                prev_char = pot_char;
            }
            line[line_iter] = '\0'; /* adding sentinel */

            /* stripping whitespace */
            if (line[line_iter - 1] == CH_BLANK) {
                line[line_iter - 1] = '\0';
            }
        return run_again;
        }

    /*****************************************************************************/

    /* algorithms are fun */
```