```
================================================================================
            /local/submit/submit/comp10002/ass1.late/chanjieh/src/ass.c
================================================================================

/* Solution to comp10002 Assignment 1, 2018 semester 2.

 * Authorship Declaration:

 * 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 further
 * 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.

 * I further 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.

 * Signed by: [Chan Jie Ho @@@ 961948]
 * Dated:     [1/9/18]

 */


/* ============================================================================ */

/* Libraries to include and hash-defined variables sorted alphabetically */

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

#define GNU_SOURCE

#define EMPTY               0
#define ERROR              -1
#define FIRST               0       /* MAY BE CHARACTER OR STRING */
#define FIRST_TEN           10
#define MAX_FRAGMENTS       1000
#define MAX_STRING_LENGTH   20
#define MULTIPLE_OF_FIVE    5
#define NO                  0
#define NON_EMPTY           1
#define NULL_BYTE           1
#define SECOND              1       /* MAY BE CHARACTER OR STRING */
#define STAGE_ONE           1
#define STAGE_THREE         3
#define STAGE_TWO           2
#define STAGE_ZERO          0
#define YES                 1
#define ZERO_OFFSET         1


/* ============================================================================ */

/* Function prototypes */

typedef char fragment_t[MAX_FRAGMENTS + NULL_BYTE];

int mygetchar(void);

char *mystrcasestr(char superstring[], fragment_t fragment);

int get_fragments(char fragment[], int limit, int *characters);
```

```
75
    void stage1(char superstring[], fragment_t new_frags[], int frags) ;

    void stage2(char superstring[], fragment_t new_frags[], int frags);

80  void stage3(char superstring[], fragment_t new_frags[], int frags);

    void initialise(char  superstring[], fragment_t new_frags[]) ;

    int find_within(char superstring[], fragment_t new_frags[], int frags,
85  int* position);

    void capital(char superstring[], fragment_t new_frags[], int index,
    int position, int i);

90  int find_overlap(char superstring[], char fragment[], int *overlap);

    void append(char superstring[], char fragment[], int index);

    void output(char superstring[], int output, int frag);
95

    /* ============================================================================ */

    /* Main function */

100
    int main(int argc, char *argv[]) {
        fragment_t one_frag, all_frags[MAX_FRAGMENTS], new_frags[MAX_FRAGMENTS];
        int stage, sum, frags = FIRST, i = FIRST;
        char superstring[MAX_FRAGMENTS * MAX_STRING_LENGTH + NULL_BYTE];
105
        /* Iterate through the input file to copy fragments into array and
         * incrementing frags to keep count of how many fragments are present
         * while keeping count of the number of characters
         */
110
        while ((get_fragments(one_frag, MAX_STRING_LENGTH, &sum)) != EOF) {
            strcpy(all_frags[i++], one_frag);
            frags++;
        }
115
        /* Iterate through stages 0 to 3 and print the output header */

        for (stage = STAGE_ZERO ; stage <= STAGE_THREE ; stage++) {
            printf("\nStage %d Output \n--------------\n", stage);
120
            /* First create new array of fragments that can be altered without
             * changing the original
             */

125         for (i = FIRST ; i < frags ; i++) {
                strcpy(new_frags[i], all_frags[i]);
            }

            if (stage == STAGE_ZERO) {
130             printf("%d fragments read, %d characters in total\n\n", frags, sum);
            }

            else if (stage == STAGE_ONE) {
                stage1(superstring, new_frags, frags);
135         }

            else if (stage == STAGE_TWO) {
                stage2(superstring, new_frags, frags);
            }
140
            else {
                stage3(superstring, new_frags, frags);
            }
        }
145
        return 0;
    }
```

```
/* ========================================================================== */

/* Helper functions by order of appearance */

/* Function written by Allistair Moffat to avoid problems with using getchar()
 */

int mygetchar(void) {
    int c;

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


/* -------------------------------------------------------------------------- */

/* Function written by Allistair Moffat to avoid problems when using
 * strcasestr() with variable names changed to ones that relate to the
 * assignment more
 */

char *mystrcasestr(char superstring[], fragment_t fragment) {
    int super_length = strlen(superstring);
    int frag_length = strlen(fragment);
    int i;

    for (i = FIRST ; i <= super_length - frag_length ; i++) {
        if (strncasecmp(superstring + i, fragment, frag_length) == NO) {
        return superstring+i;
        }
    }
    return NULL;
}

/* -------------------------------------------------------------------------- */

/* Iterate through every character and store each line of characters as a
 * fragment
 */

int get_fragments(char fragment[], int limit, int *characters) {
    int c, length = EMPTY;

    /* Upon reaching the end of the file */

    if ((c = mygetchar()) == EOF) {
        return EOF;
    }

    fragment[length++] = c;
    while ((length < limit) && (((c = mygetchar()) != EOF) && (c != '\n'))) {
        fragment[length++] = c;
    }

    /* Add the zero character at the end of each fragment to make it a string
     * and increment the total number of characters by the length of each
     * string
     */

    fragment[length] = '\0';
    *characters += length;
    return 0;
}

/* Function reworked from getwords.c (created by Allistair Moffat) */

/* ============================================================================
   Program written by Alistair Moffat, as an example for the book
   "Programming, Problem Solving, and Abstraction with C", Pearson
   Custom Books, Sydney, Australia, 2002; revised edition 2012,
```

```
       ISBN 9781486010974.

225    See http://people.eng.unimelb.edu.au/ammoffat/ppsaa/ for further
       information.

       Prepared December 2012 for the Revised Edition.
       ========================================================================= */

230

   /* ------------------------------------------------------------------------- */

   /* Stage 1 */

235
   void stage1(char superstring[], fragment_t new_frags[], int frags) {
       int frag_length, index, overlap, i, super_length;
       char *sub;

240    /* Initialise superstring as the first fragment */

       initialise(superstring, new_frags);

       for (i=SECOND ; i<frags ; i++) {
245
           frag_length = strlen(new_frags[i]);
           super_length = strlen(superstring);

           /* Check if the fragment is already within the superstring */
250
           sub = mystrcasestr(superstring, new_frags[i]);

           if (sub == NULL) {

255            /* Doing this means it was not found so we find if there is any
                * overlap at the end of the superstring
                */

               index = find_overlap(superstring, new_frags[i], &overlap);
260
               if (index < super_length) {

                   /* Doing this means there was an overlap so append the fragment
                    * at the end of the superstring
265                 */

                   append(superstring, new_frags[i], index);

               }
270            else {

                   /* Capitalise the first character of the fragment and append it
                    * at the end of the superstring
                    */
275
                   new_frags[i][FIRST] = toupper(new_frags[i][FIRST]);
                   strcat(superstring, new_frags[i]);
               }
           }
280        else {

               /* Capitalise the first letter where the fragment can be found */
               index = sub - &superstring[FIRST];
               superstring[index] = toupper(superstring[index]);
285        }

           super_length = strlen(superstring);
           if (i <= FIRST_TEN || (i%MULTIPLE_OF_FIVE)==EMPTY) {
               printf("%d: frg= %d, slen= %d  %s\n", i, i, super_length,
290            superstring);
           }
       }
       printf("---\n%d: frg=-1, slen= %d  %s\n", i - ZERO_OFFSET, super_length,
       superstring);
295 }
```

```
/* ------------------------------------------------------------------------ */

/* Stage 2 */

void stage2(char superstring[], fragment_t new_frags[], int frags) {
    int i, j, index, position, overlap, max_overlap, frg, frg_index;
    char fragment[MAX_STRING_LENGTH+NULL_BYTE];

    /* Initialise superstring as the first fragment */

    initialise(superstring, new_frags);
    for (i = SECOND ; i < frags ; i++) {
        position = EMPTY;

        /* Find if any fragments are within the superstring */

        index = find_within(superstring, new_frags, frags, &position);
        if (index > EMPTY) {

            /* Capitalise first letter of the first fragment found in the
             * superstring
             */

            capital(superstring, new_frags, index, position, i);
        }

        else {

            /* Find the max overlap */
            max_overlap = ERROR;
            frg = frags;

            for (j = SECOND ; j < frags ; j++) {

                strcpy(fragment, new_frags[j]);

                /* Find the number of characters that overlap */

                index = find_overlap(superstring, fragment, &overlap);

                if ((overlap > max_overlap) && (strlen(fragment)) > NON_EMPTY) {

                    /* New max overlap found so make note of the fragment
                     * number
                     */

                    max_overlap = overlap;
                    frg = j;
                    frg_index = index;
                }
            }

            /* Append the fragment with the largest overlap and make it a zero
             * character to be marked as processed
             */

            append(superstring, new_frags[frg], frg_index);
            strcpy(new_frags[frg], "\0");
            output(superstring, i, frg);
        }
    }
    printf("---\n%d: frg=-1, slen= %lu  %s\n", i - ZERO_OFFSET,
    strlen(superstring), superstring);
}


/* ------------------------------------------------------------------------ */

/* Stage 3 */

void stage3(char superstring[], fragment_t new_frags[], int frags) {
    int i, j, index, position, max_overlap, frg, frg_index;
    int ap_index, pre_index, ap_overlap, pre_overlap, max, prepend;
```

```
        char fragment[MAX_STRING_LENGTH+NULL_BYTE];

        initialise(superstring, new_frags);

375     /* Same as stage 2 for the most part, comments where different */

        for (i = SECOND ; i < frags ; i++) {
            position = EMPTY;
            index = find_within(superstring, new_frags, frags, &position);
380         if (index > EMPTY) {
                capital(superstring, new_frags, index, position, i);
            }
            else {
                max_overlap = ERROR;
385             frg = frags;
                for (j = SECOND ; j < frags ; j++) {
                    strcpy(fragment, new_frags[j]);

                    /* Get the number of characters that a fragment overlaps at
390                  * the end and then flip it around to get the number of
                     * character the fragment overlaps at the beginning
                     */

                    ap_index = find_overlap(superstring, fragment, &ap_overlap);
395                 pre_index = find_overlap(fragment, superstring, &pre_overlap);

                    /* Choose the max overlap out of the two options */

                    max = ap_overlap;
400                 if (pre_overlap > max) {
                        max = pre_overlap;
                    }

                    if ((max > max_overlap) && (strlen(fragment)) > NON_EMPTY) {
405
                        max_overlap = max;
                        frg = j;
                        frg_index = pre_index;

410                     /* Decide if prepend or not */

                        prepend = YES;
                        if (max == ap_overlap) {
                            frg_index = ap_index;
415                         prepend = NO;
                        }
                    }
                }

420             if (prepend == NO) {

                    /* Append like normal */

                    append(superstring, new_frags[frg], frg_index);
425             }
                else {

                    /* Prepend instead */

430                 append(new_frags[frg], superstring, frg_index);
                    strcpy(superstring,new_frags[frg]);
                    superstring[FIRST] = toupper(superstring[FIRST]);
                }

435             strcpy(new_frags[frg], "\0");
                output(superstring, i, frg);

            }
        }
440     printf("---\n%d: frg=-1, slen= %lu  %s\n", i - ZERO_OFFSET,
        strlen(superstring), superstring);
    }
```

```
/* ------------------------------------------------------------------------ */

/* Initialise first fragment as the first input of the superstring and
 * capitalise the first letter then make that fragment empty
 */

void initialise(char  superstring[], fragment_t new_frags[]) {
    strcpy(superstring, new_frags[FIRST]);
    superstring[FIRST] = toupper(superstring[FIRST]);
    printf("0: frg= 0, slen= %lu  %s\n", strlen(superstring), superstring);
    strcpy(new_frags[FIRST], "\0");
}


/* ------------------------------------------------------------------------ */

/* Find the first fragment already present in the superstring and returns the
 * fragment number or an error (negative number) if none are found
 */

int find_within(char superstring[], fragment_t fragment[], int total_fragments,
int *position) {
    int i;
    char *index;

    for (i=SECOND ; i < total_fragments ; i++) {

        index = mystrcasestr(superstring, fragment[i]) ;
        if ((index != NULL) && isalpha(*fragment[i])) {

            /* Doing this means a fragment can be found within the superstring
             * and is not just a zero character so return the fragment number
             * and the position of the first character
             */

            *position = index - &superstring[FIRST];
            return i;
        }
    }
    return ERROR;
}


/* ------------------------------------------------------------------------ */

/* Capitalise the first letter of the fragment found within the superstring */

void capital(char superstring[], fragment_t new_frags[], int index,
int position, int i) {
    int length;

    length = strlen(superstring);

    /* Make the fragment empty to show that it has been processed */

    strcpy(new_frags[index], "\0");
    superstring[position] = toupper(superstring[position]);

    /* Print only if the output is the first ten or a multiple of five */

    if (i <= FIRST_TEN || (i%MULTIPLE_OF_FIVE) == EMPTY) {
        printf("%d: frg= %d, slen= %d  %s\n", i, index, length, superstring);
    }
}


/* ------------------------------------------------------------------------ */

/* Code to find if any fragments are overlapping with the superstring and
 * returns the position of the which the fragment overlaps, or returns the
 * length of the superstring if it is not overlapping
 */

int find_overlap(char superstring[], fragment_t fragment, int *overlap) {
```

```
        char partial[MAX_FRAGMENTS * MAX_STRING_LENGTH + NULL_BYTE];
520     char super[MAX_FRAGMENTS * MAX_STRING_LENGTH + NULL_BYTE];
        char *index;
        int frag_length = EMPTY, super_length, position, i, j;

        /* Check if the fragment can be found within the supestring */
525
        index = mystrcasestr(superstring, fragment);
        if (index == NULL) {

            /* Check if part of the fragment can be found by comparing against the
530          * first (length of fragment minus i) characters of the fragment
             */

            for (i=FIRST ; i < strlen(fragment) ; i++) {

535             /* As i increases, length of partial fragment will decrease */

                frag_length = strlen(fragment) - i;
                super_length = strlen(superstring);

540             /* Create new array to hold partial fragment that can be edited
                 * without altering the original fragment and then add a zero
                 * character at the end
                 */

545             for (j=FIRST ; j < frag_length ; j++) {
                    partial[j] = fragment[j];
                }
                partial[j] = '\0';

550             /* Same for the superstring but get the last (length of partial fragmen
    t) characters instead */

                for (j=FIRST ; j < frag_length ; j++) {
                    super[j] = superstring[super_length - frag_length + j];
                }
555             super[j] = '\0';

                /* Check if partial fragment and partial superstring are the same */

                index = mystrcasestr(super, partial);
560
                if (index != NULL) {

                    /* Doing this means the fragment overlaps so give the position
                     * of the first overlapping character and get out of the loop
565                  */

                    position = super_length - frag_length;
                    break;
                }
                else {
570                 index = "Not found";
                }
            }
        }
575
        else {

            /* Give position of where the fragment can be found */

580         position = index - superstring;
        }

        if (mystrcasestr(index, "Not found") != NULL) {

585         /* Doing this means the fragment does not overlap at all so return the
             * length of the superstring
             */

            *overlap = EMPTY;
590         return super_length;
        }
```

```
        /* Return position of the first overlapping character */

595     *overlap = frag_length;
        return position;
    }


600 /* ---------------------------------------------------------------------- */

    /* Append fragment into the superstring at the position that the overlap starts
     */

605 void append(char superstring[], fragment_t fragment, int index) {
        int i;
        char super[MAX_FRAGMENTS * MAX_STRING_LENGTH + NULL_BYTE];

        /* Create a new array to hold the part of the superstring before the
610      * overlap and add a zero character after it and capitalise the first letter
         */

        for (i=FIRST ; i < index ; i++) {
            super[i] = superstring[i];
615     }

        super[i] = '\0';
        fragment[FIRST] = toupper(fragment[FIRST]);

620     /* Append the fragment to the partial superstring and replace the original
         * superstring with end result
         */

        strcat(super, fragment);
625     strcpy(superstring, super);
    }


    /* ---------------------------------------------------------------------- */
630
    /* Output function that prints the first ten outputs and one every five output
     * after to cut down on repetition of code in stages */

    void output(char superstring[], int output, int frag) {
635     int length;

        length = strlen(superstring);
        if (output <= FIRST_TEN || (output%MULTIPLE_OF_FIVE)==EMPTY) {
            printf("%d: frg= %d, slen= %d  %s\n", output, frag, length,
640         superstring);
        }
    }


645 /* ====================================================================== */

    /* aLgOrItHmS aRe FuN */
```