```cpp
/*
 Jingxian Liu & Qi (Sara) Zhang
 Program Name: Project OS2

 Purpose: The purpose of this project is to use both threads and semaphonres
    to simulate the Latex text formating system.

 Input:
    1) Input from keyboard: Filename
    2) An input text file of any length, which contains normal words,
        punctuations, and control sequences '\c' (capitalize the first
        letter of the next word), '\C' (capitalize all of the letters
        of the next word), and '\u' (underline the next word (simulate
        this by _word_))

 Output:
    1) Output on console:
            Prompt to enter input filename
            Error messages while file not found
    2) Output files: finalAnswer.txt

 - We have abided by the Wheaton College honor code in this work.
*/

#include <iostream>
#include <fstream>
#include <sstream>
#include <unistd.h>
#include <cstdlib> // Avoid the error related to exit() function
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>


using namespace std;

// Function declarations
void * Store_Char_Array(void *);
void * Process_One_Line(void *);
void * Align_Display_Text(void *);

// Declare global constants and variables
const int total_lines = 200;
const int total_chars = 60;
const int outcolumn=50;
pthread_mutex_t mutex1, mutex2, mutex3;  // Counter semaphores

// Declare global boolean values
bool threadOneDone = false;
bool threadTwoDone = false;

int main() {

    // Initialize a two dimensional array with 200 lines,
    // and 60 characters per line
    char ** input_content = new char*[total_lines];
    for (int i = 0; i < total_lines; i++) {
        input_content[i] = new char[total_chars];
    }

    // Set up semaphores
    pthread_mutex_init (&mutex1, NULL);
    pthread_mutex_init (&mutex2, NULL);
```

```
65      pthread_mutex_init (&mutex3, NULL);
66
67      // Lock mutex2 and mutex3
68      pthread_mutex_lock (&mutex2);
69      pthread_mutex_lock (&mutex3);
70
71      // Initialize and create three processes
72      pthread_t thread1, thread2, thread3;
73      pthread_create(&thread1, NULL, Store_Char_Array, (void *) input_content);
74      pthread_create(&thread2, NULL, Process_One_Line, (void *) input_content);
75      pthread_create(&thread3, NULL, Align_Display_Text, (void *) input_content);
76
77      // Wait for each thread to finish
78      pthread_join(thread1, NULL);
79      pthread_join(thread2, NULL);
80      pthread_join(thread3, NULL);
81
82      // Deallocate semaphores
83      pthread_mutex_destroy (&mutex1);
84      pthread_mutex_destroy (&mutex2);
85      pthread_mutex_destroy (&mutex3);
86
87      return 0;
88  }
89
90
91  void * Store_Char_Array(void * ptr)
92  {
93      /* This function takes value from an input file and store the value into a two
    dimensional array, which has 200 lines in total, and 60 characters per line.
94       - Pre-condition: An character array pointer 'ptr' needs to be passed into this
    function. 'ptr' points to a two dimensional array.
95       - Post-condition: The modified 'ptr' will be passed back to the main function by
    reference.
96       - Return: None
97       */
98
99
100     // Pass the pointer (pass by reference), and then cast void pointer to struct
    storeSublist
101     char **temp_ptr = (char **) ptr;
102
103     // Declare and set input filename
104     string filename;
105
106     // Get input filename from keyboard
107     cout << "Please enter filename: (e.g., randomNums.txt)\n";
108     getline(cin, filename);
109
110     cout << "Filename: " << filename << endl;
111
112     // Open input file. Exit if cannot open the file, print error message on console
113     ifstream input_file;
114     input_file.open(filename.c_str());
115     if (!input_file.is_open())
116     {
117         cout<<"Error, file does not exist";
118         exit(-1);
119     }
120
121
122     int count_line = 0;
123     string next_line;
124     // Iterate through each line of the input file to store data into the two dimensional
    array
```

```
125     while (getline(input_file, next_line)) {
126         // Lock itself
127         pthread_mutex_lock(&mutex1);
128 
129         for(int i=0; i<next_line.length(); i++){
130             temp_ptr[count_line][i] = next_line[i];
131         }
132 
133         count_line++;
134 
135         // Unlock thread 2
136         pthread_mutex_unlock(&mutex2);
137 
138     }
139 
140     // Change the boolean value after the first thread finished
141     threadOneDone = true;
142 }
143 
144 
145 void * Process_One_Line(void * ptr)
146 {
147     /* This function handles each line of the two dimensional array, and changes the characters based on the rules of the control sequences.
148      - Pre-condition: An character array pointer 'ptr' needs to be passed into this function. 'ptr' points to a two dimensional array.
149      - Post-condition: The modified 'ptr' will be passed back to the main function by reference.
150      - Return: None
151      */
152 
153     // Pass the pointer (pass by reference), and then cast void pointer to struct storeSublist
154     char **temp_ptr = (char **) ptr;
155 
156     int j=0, i=0;
157     // Running thread 2 while the first thread has not finished running
158     while (!threadOneDone) {
159         // Lock itself
160         pthread_mutex_lock(&mutex2);
161 
162         j=0;
163         while (temp_ptr[i][j]!= 0){
164             if (temp_ptr[i][j]=='\\'){
165                 // Capitalize the first letter of the next word if the word is led by \c
166                 if (temp_ptr[i][j+1]=='c'){
167                     temp_ptr[i][j+2] = toupper(temp_ptr[i][j+2]);
168                     temp_ptr[i][j] = ' ';
169                     temp_ptr[i][j+1] =' ';
170                 }
171                 // Capitalize all of the letters of the next word if the word is led by \C
172                 else if (temp_ptr[i][j+1]=='C'){
173 
174                     int temp = j;
175                     while(temp_ptr[i][temp+2] != ' ' && temp_ptr[i][temp+2] != 0){
176                         temp_ptr[i][temp+2] = toupper(temp_ptr[i][temp+2]);
177                         temp++;
178                     }
179                     temp_ptr[i][j] = ' ';
180                     temp_ptr[i][j+1] = ' ';
181 
182                 }
183                 // Underline the next word (simulate this by _word_) if the word is led by \u
184                 else if(temp_ptr[i][j+1]=='u'){
```

```
185                     temp_ptr[i][j] = '_';
186                     int temp = j;
187                     while(temp_ptr[i][temp+1]!=' ' && temp_ptr[i][temp+1] != '\n'){
188                         temp_ptr[i][temp+1] = temp_ptr[i][temp+2];
189                         temp++;
190                     }
191                     temp_ptr[i][temp] = '_';
192                 }
193             }
194             j++;
195         }
196         i++;
197 
198         // Unlock thread 3
199         pthread_mutex_unlock(&mutex3);
200 
201     }
202 
203     // Change the boolean value after the second thread finished
204     threadTwoDone = true;
205 
206 }
207 
208 
209 void * Align_Display_Text(void * ptr)
210 {
211     /* This function takes each of the modified line and displays the text on both the
    console and the screen. The output is left and right justified.
212      - Pre-condition: An character array pointer 'ptr' needs to be passed into this
    function. 'ptr' points to a two dimensional array.
213      - Post-condition: The modified 'ptr' will be passed back to the main function by
    reference.
214      - Return: None
215      */
216 
217     // Pass the pointer (pass by reference), and then cast void pointer to struct
    storeSublist
218     char ** temp_ptr = (char **) ptr;
219 
220     // Declare output stream
221     ofstream outfile;
222     // Generate the output into an output file called "finalAnswer.txt"
223     outfile.open("finalAnswer.txt");
224 
225     int i = 0;
226     // Running thread 3 while the first and the second threads have not finished running
227     while (!threadOneDone || !threadTwoDone) {
228         // Lock itself
229         pthread_mutex_lock (&mutex3);
230         //count the length of the row
231         int countnum = 0;
232         while(temp_ptr[i][countnum] != 0){
233             countnum++;
234         }
235         int index = 0;
236         // check if both thread one and two are done
237         // if it is, just print out the last line
238         if (threadOneDone && threadTwoDone){
239             for (int j =0; j< countnum; j++){
240                 outfile << temp_ptr[i][j];
241                 cout << temp_ptr[i][j];
242             }
243             outfile << endl;
244             cout << endl;
245             outfile.close();
```

```
246
247         }
248         //if not, put space into the last space
249         else{
250
251             //get the space in the row
252
253             for(int j = 0; j < countnum; j++){
254                 if(temp_ptr[i][j] == ' '){
255                     index = j;
256                 }
257             }
258             //insert in spaces where there are space to fill 50 slot
259
260             for(int j = 0; j < index; j++){
261
262                 cout << temp_ptr[i][j];
263                 outfile << temp_ptr[i][j];
264             }
265
266             for(int k = index; k < (outcolumn - countnum + index); k++){
267
268                 cout << ' ';
269                 outfile << ' ';
270             }
271
272             for(int l = (outcolumn - countnum + index); l < outcolumn; l++){
273                 //print out the rest of the row
274                 cout << temp_ptr[i][l-(50 - countnum)];
275                 outfile << temp_ptr[i][l-(outcolumn - countnum)];
276             }
277
278             i++;
279
280             // Unlock thread 1
281             pthread_mutex_unlock (&mutex1);
282             cout << endl;
283             outfile << endl;
284         }
285     }
286
287 }
```