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


    /* Error Code List */
    /*  no error = 0
        duplicate labels = 1
        illegal label = 2
        illegal operation = 3
        missing data directive = 4
        missing operan start = 5
        mising opcode end = 6
        too many symbols = 7
        program too long = 8 */

typedef struct
{
    char label[10];
    int memoryAddress;
}LABELS;

typedef struct
{
    char *label;
    char *mnemonic;
    char *opcode;
} TOKEN;

typedef struct
{
    char mnemonic[5];
    int opcode;
} OPCODE;

void breakupLine(char *input, char *command, char *param1, char *param2, int *numParams);
void loadFile(char *fileName);
void passOne(char * fileName);
void passTwo();
void executeFile();
void debugFile();
void dumpFile();
void helpFile();
void assembleFile();
void errorFile();
int programLenght;

```

```c
int main(void)
{
    char input[50];
    char command[50];
    char param1[50];
    char param2[50];

    printf("Hello welcome to Jesus Morales Personal Assembler\n \n");
    while (1)
    {
        int numParams = 0;
        int len = 0;
        printf("Command ----> ");
        fgets(input, 50, stdin);

        len = strlen(input) - 1;
        if (input[len] == '\n')
        {
            input[len] = '\0';
        }

        breakupLine(input, command, param1, param2, &numParams);
        numParams--;

        if (strcmp(command, "load") == 0)
        {
            if (numParams == 1)
            {
                loadFile(param1);
            }
            else
                errorFile();
        }
        else if (strcmp(command, "execute") == 0)
        {
            executeFile();
        }
        else if (strcmp(command, "debug") == 0)
        {
            debugFile();
        }
        else if (strcmp(command, "dump") == 0)
        {
            if (numParams == 2)
            {
                dumpFile();
            }
            else
                errorFile();
```

```
 98          }
 99          else if (strcmp(command, "help") == 0)
100          {
101              helpFile();
102          }
103          else if (strcmp(command, "assemble") == 0)
104          {
105              if (numParams == 1)
106              {
107                  assembleFile();
108              }
109              else
110                  errorFile();
111          }
112          else if (strcmp(command, "dir") == 0)
113          {
114              system("dir");
115          }
116          else if (strcmp(command, "exit") == 0)
117          {
118              break;
119          }
120          else
121          {
122              printf("Invalid Command , for any help type 'help' to display the
                   command list. \n \n");
123          }
124
125          numParams = 0;
126      }
127      return 0;
128  }
129
130  void breakupLine(char *input, char *command, char *param1, char *param2, int
       *numParams)
131  {
132      command[0] = param1[0] = param2[0] = '\0';
133      *numParams = sscanf(input, "%s %s %s %*s", command, param1, param2);
134  }
135
136  void loadFile(char *param1)
137  {
138      printf("Loading file:  %s\n", param1);
139      passOne(param1);
140      passTwo();
141      printf("The Programg lenght of this file is:   %d Bytes\n\n", programLenght);
142      programLenght = 0;
143
144  }
```

```c
void executeFile()
{
    printf(" is not yet avaibalbe.\n");
}
void debugFile()
{
    printf("debug is not avaialabe.\n");
}
void dumpFile()
{
    printf("dump is not avaiblable.\n");
}
void helpFile()
{
    printf("\n");
    printf("\tWelcome to the Help menu. \n");
    printf("\tCommands are the following: \n \n");
    printf("\tload [file_name]\n");
    printf("\texecute \n");
    printf("\tdebug \n");
    printf("\tdump [start] [end] \n");
    printf("\thelp \n");
    printf("\tassemble [file_name] \n");
    printf("\tdirectory \n");
    printf("\texit \n\n");
    printf("\t**ALL COMMANDS ARE CASE SENSITIVE.**\n\n");
}
void assembleFile()
{
    printf("assemble not avaibalbe. \n");
}
void errorFile()
{
    printf("You typed the wrong number of parameters try again. \n");
}
void passOne(char *param1)
{
    char input[500];
    char *tokenizer = input;

    int start = 0;
    int locctr = 0;
    int memLenght = 0;
    int numLabels = 0;
    int numMnemonics = 25;
    int index = 0;

    int labelPresentFlag = 0;
    int duplicateLabelFlag = 0;
```

```
194     int illegalLabelFlag = 0;
195     int illegalOperationFlag = 0;
196     int missingDataDirectiveFlag = 0;
197     int missingStartFlag = 0;
198     int missingEndFlag = 0;
199     int tooManyLabelsFlag = 0;
200     int programTooLongFlag = 0;
201     int errorCode = 0;
202
203     FILE *source_file, *symbol_file, *intermediate_file, *opcode_file;
204     LABELS labelStructure[500];
205     TOKEN tokenStructure;
206     OPCODE opcodeStructure[] = { { "ADD", 0x18 },{ "AND",0x58 },{ "COMP", 0x28 },
        { "DIV", 0x24 },
207                                  { "J", 0x3C },{ "JEQ", 0x30 },{ "JGT", 0x34 },
                    { "JLT", 0x38 },
208                                  { "JSUB", 0x48 },{ "LDA", 0x00 },{ "LDCH",
                    0x50 },{ "LDL", 0x08 },
209                                  { "LDX", 0x04 },{ "MUL", 0x20 },{ "OR", 0x44 },
                    { "RD", 0xD8 },
210                                  { "RSUB", 0x4C },{ "STA", 0x0C },{ "STCH",
                    0x54 },{ "STL", 0x14 },
211                                  { "STX", 0x10 },{ "SUB", 0x1C },{ "TD", 0xE0 },
                    { "TIX", 0x2C },{ "WD", 0xDC } };
212
213     source_file = fopen(param1, "r");
214     intermediate_file = fopen("intermediate.txt", "w");
215     symbol_file = fopen("symbolTable.txt", "w");
216
217     if (source_file == NULL)
218     {
219         printf("Error openning file does not exist: %s\n", param1);
220         return;
221     }
222
223     tokenStructure.label = (char *)malloc(6);
224     tokenStructure.mnemonic = (char *)malloc(6);
225     tokenStructure.opcode = (char *)malloc(6);
226
227     while (fgets(input, 500, source_file))
228     {
229         labelPresentFlag = 0;
230         duplicateLabelFlag = 0;
231         illegalLabelFlag = 0;
232         illegalOperationFlag = 0;
233         missingDataDirectiveFlag = 0;
234         missingStartFlag = 0;
235         missingEndFlag = 0;
236         tooManyLabelsFlag = 0;
```

```
237         programTooLongFlag = 0;
238         errorCode = 0;
239         memLenght = 0;
240 
241         /*  Check if label is present in the string line    */
242         if (input[0] == ' ' || input[0] == '\t')
243         {
244             labelPresentFlag = 0;
245         }
246         else
247         {
248             labelPresentFlag = 1;
249         }
250 
251         /*  Check if comment is present in the string line  */
252         if (input[0] == '.')
253         {
254             continue;
255         }
256 
257         /*  Tokenize the input string   */
258         tokenizer = strtok(input, " \t\r\n\v\f");
259 
260         /*  Remove of the trailing newLine at the end of the string */
261         int counter = 0;
262         while (input[counter - 1] != '\n')
263         {
264             counter++;
265         }
266         input[counter] = '\0';
267 
268         /*  If there is a label */
269         if (labelPresentFlag == 1)
270         {
271             /*  Tokenize the label into the structure   */
272             strcpy(tokenStructure.label, tokenizer);
273 
274             /*  Tokenize the mnemonic into the structure    */
275             tokenizer = strtok(NULL, " \t\r\n\v\f");
276             strcpy(tokenStructure.mnemonic, tokenizer);
277 
278             /*  Tokenize the opcode of the mnemonic into the structure  */
279             tokenizer = strtok(NULL, " \t\r\n\v\f");
280             strcpy(tokenStructure.opcode, tokenizer);
281 
282             /*  Add the labels to the structure to create a list of existing
                  labels/symbols */
283             strcpy(labelStructure[numLabels].label, tokenStructure.label);
284             labelStructure[numLabels].memoryAddress = locctr;
```

```
285
286             /*  Check if there are labels in the list   */
287             if (numLabels > 0)
288             {
289                 /*  Check if limit of labels has been reached   */
290                 if (numLabels > 500)
291                 {
292                     tooManyLabelsFlag = 1;
293                 }
294
295                 /*  Inefficiently scan the label/symbol list to check for
                      duplicate labels/symbols     */
296                 for (int i = 0; i < numLabels; i++)
297                 {
298                     if (strcmp(labelStructure[i].label, tokenStructure.label) ==
                       0)
299                     {
300                         duplicateLabelFlag = 1;
301                     }
302                 }
303             }
304
305             /*  Check if the label is legal */
306             if (!isalpha(tokenStructure.label[0]))
307             {
308                 illegalLabelFlag = 1;
309             }
310
311             /*  Check if we have a START directive in the beginning of the
                  program */
312             if (index == 0 && strcmp(tokenStructure.mnemonic, "START") != 0)
313             {
314                 missingStartFlag = 1;
315                 locctr = 0;
316             }
317
318             /*  Check if we have a END directive in the end of the program  */
319             if (missingEndFlag == 1 && errorCode == 0)
320             {
321                 if (strcmp(tokenStructure.mnemonic, "END") != 0)
322                 {
323                     missingEndFlag = 1;
324                 }
325             }
326
327             /*  If directive START initialize LOCCTR to the starting address
                  */
328             if (strcmp(tokenStructure.mnemonic, "START") == 0) // if start
                  directive initialize locct to the start(convert the string to
```

```
                integer)
329             {
330                 start = atoi(tokenStructure.opcode);
331                 locctr = start;
332             }
333
334             /*  Check if program is too long    */
335             if (locctr > 6700)
336             {
337                 programTooLongFlag = 1;
338             }
339             /* Lenght size in memory from the directives to increment the LOCCTR
                    */
340             if (strcmp(tokenStructure.mnemonic, "WORD") == 0)
341             {
342                 memLenght += 3;
343             }
344             if (strcmp(tokenStructure.mnemonic, "RESB") == 0)
345             {
346                 memLenght += atoi(tokenStructure.opcode);
347
348             }
349             if (strcmp(tokenStructure.mnemonic, "RESW") == 0)
350             {
351                 memLenght += 3 * atoi(tokenStructure.opcode);
352             }
353
354             if (strcmp(tokenStructure.mnemonic, "BYTE") == 0)
355             {
356                 /*  Check if operand is set to read a string (C) or a hexadecimal
                    (X)    */
357                 if (tokenStructure.opcode[0] == 'C')
358                 {
359
360                     int bufferSpace = 0;
361                     int counter = 2;
362                     while (tokenStructure.opcode[counter] != '\'' && bufferSpace
                     < 30)
363                     {
364                         bufferSpace++;
365                         counter++;
366                     }
367                     memLenght += bufferSpace;
368
369                 }
370                 else if (tokenStructure.opcode[0] == 'X')
371                 {
372                     char hexInput[16];
373                     int bufferSpace = 0;
```

```
374                     int counter = 2;
375                     while (tokenStructure.opcode[counter] != '\'' && bufferSpace
                         < 16)
376                     {
377                         hexInput[bufferSpace] = tokenStructure.opcode[counter];
378                         bufferSpace++;
379                         counter++;
380                     }
381                     hexInput[bufferSpace] = '\0';
382                     memLenght = (int)strtol(hexInput, NULL, 16);
383                 }
384 
385                 /* Check for errors in the input for the BYTE directive */
386                 else
387                 {
388                     illegalOperationFlag = 1;
389                 }
390 
391                 if (tokenStructure.opcode[1] != '\'' || tokenStructure.opcode
                    [strlen(tokenStructure.opcode) - 1] != '\'')
392                 {
393                     missingDataDirectiveFlag = 1;
394                 }
395             }
396 
397             /* Error Flag conditions    */
398             if (duplicateLabelFlag == 1 && errorCode == 0)
399             {
400                 errorCode = 1;
401             }
402             else if (illegalLabelFlag == 1 && errorCode == 0)
403             {
404                 errorCode = 2;
405             }
406             else if (illegalOperationFlag == 1 && errorCode == 0)
407             {
408                 errorCode = 3;
409             }
410             else if (missingDataDirectiveFlag == 1 && errorCode == 0)
411             {
412                 errorCode = 4;
413             }
414             else if (missingStartFlag == 1 && errorCode == 0)
415             {
416                 errorCode = 5;
417             }
418             else if (missingEndFlag == 1 && errorCode == 0)
419             {
420                 errorCode = 6;
```

```
421             }
422             else if (tooManyLabelsFlag == 1 && errorCode == 0)
423             {
424                 errorCode = 7;
425             }
426             else if (programTooLongFlag == 1 && errorCode == 0)
427             {
428                 errorCode = 8;
429             }
430 
431             /*Print to the intermediate file and symbol file */
432             fprintf(intermediate_file, "%d\t%s\t%s\t%s\t%d\n", locctr,
                  tokenStructure.label, tokenStructure.mnemonic,
                  tokenStructure.opcode, errorCode);
433             fprintf(symbol_file, "%d\t %s\n", locctr, tokenStructure.label);
434 
435             /* Search for the mnemonic in the opcode table and add 3 to it  */
436             for (int i = 0; i < numMnemonics; i++)
437             {
438                 if (strcmp(opcodeStructure[i].mnemonic, tokenStructure.mnemonic)
                      == 0)
439                 {
440                     locctr += 3;
441                 }
442             }
443 
444             /*Update the memory locations after LOCCTR is printed in the file
                  */
445             if (strcmp(tokenStructure.mnemonic, "BYTE") == 0 || strcmp
                  (tokenStructure.mnemonic, "RESB") == 0 || strcmp
                  (tokenStructure.mnemonic, "RESW") == 0 || strcmp
                  (tokenStructure.mnemonic, "WORD") == 0)
446             {
447                 locctr += memLenght;
448             }
449 
450             /*  Increment the number of labels in the system    */
451             numLabels++;
452         }
453 
454         /*If there is no label in the input line do the same as above but without
               labels */
455         else
456         {
457             /*  Tokenize the mnemonic into the structure    */
458             strcpy(tokenStructure.mnemonic, tokenizer);
459 
460             /*  Tokenize the opcode into the structure  */
461             tokenizer = strtok(NULL, " \t\r\n\v\f");
```

```
462             strcpy(tokenStructure.opcode, tokenizer);
463
464             /*  Check if we have a START directive in the beginning of the
                  program */
465             if (index == 0 && strcmp(tokenStructure.mnemonic, "START") != 0)
466             {
467                 missingStartFlag = 1;
468                 locctr = 0;
469             }
470
471             /*  Check if we have a END directive in the end of the program  */
472             if (missingEndFlag == 1 && errorCode == 0)
473             {
474                 if (strcmp(tokenStructure.mnemonic, "END") != 0)
475                 {
476                     missingEndFlag = 1;
477                 }
478             }
479
480             /*  If directive START initialize LOCCTR to the starting address
                  */
481             if (strcmp(tokenStructure.mnemonic, "START") == 0) // if start
                  directive initialize locct to the start(convert the string to
                  integer)
482             {
483                 start = atoi(tokenStructure.opcode);
484                 locctr = start;
485             }
486
487             /*  Check if program is too long    */
488             if (locctr > 6700)
489             {
490                 programTooLongFlag = 1;
491             }
492
493             /* Lenght size in memory from the directives to increment the LOCCTR
                     */
494             if (strcmp(tokenStructure.mnemonic, "WORD") == 0)
495             {
496                 memLenght += 3;
497             }
498             if (strcmp(tokenStructure.mnemonic, "RESB") == 0)
499             {
500                 memLenght += atoi(tokenStructure.opcode);
501
502             }
503             if (strcmp(tokenStructure.mnemonic, "RESW") == 0)
504             {
505                 memLenght += 3 * atoi(tokenStructure.opcode);
```

```
506             }
507
508             if (strcmp(tokenStructure.mnemonic, "BYTE") == 0)
509             {
510                 /*  Check if operand is set to read a string (C) or a hexadecimal
                       (X)    */
511                 if (tokenStructure.opcode[0] == 'C')
512                 {
513
514                     int bufferSpace = 0;
515                     int counter = 2;
516                     while (tokenStructure.opcode[counter] != '\'' && bufferSpace
                        < 30)
517                     {
518                         bufferSpace++;
519                         counter++;
520                     }
521                     memLenght += bufferSpace;
522
523                 }
524                 else if (tokenStructure.opcode[0] == 'X')
525                 {
526                     char hexInput[16];
527                     int bufferSpace = 0;
528                     int counter = 2;
529                     while (tokenStructure.opcode[counter] != '\'' && bufferSpace
                        < 16)
530                     {
531                         hexInput[bufferSpace] = tokenStructure.opcode[counter];
532                         bufferSpace++;
533                         counter++;
534                     }
535                     hexInput[bufferSpace] = '\0';
536                     memLenght = (int)strtol(hexInput, NULL, 16);
537                 }
538
539                 /* Check for errors in the input for the BYTE directive */
540                 else
541                 {
542                     illegalOperationFlag = 1;
543                 }
544
545                 if (tokenStructure.opcode[1] != '\'' || tokenStructure.opcode
                      [strlen(tokenStructure.opcode) - 1] != '\'')
546                 {
547                     missingDataDirectiveFlag = 1;
548                 }
549             }
550
```

```
551             /* Error Flag conditions    */
552             if (duplicateLabelFlag == 1 && errorCode == 0)
553             {
554                 errorCode = 1;
555             }
556             else if (illegalLabelFlag == 1 && errorCode == 0)
557             {
558                 errorCode = 2;
559             }
560             else if (illegalOperationFlag == 1 && errorCode == 0)
561             {
562                 errorCode = 3;
563             }
564             else if (missingDataDirectiveFlag == 1 && errorCode == 0)
565             {
566                 errorCode = 4;
567             }
568             else if (missingStartFlag == 1 && errorCode == 0)
569             {
570                 errorCode = 5;
571             }
572             else if (missingEndFlag == 1 && errorCode == 0)
573             {
574                 errorCode = 6;
575             }
576             else if (tooManyLabelsFlag == 1 && errorCode == 0)
577             {
578                 errorCode = 7;
579             }
580             else if (programTooLongFlag == 1 && errorCode == 0)
581             {
582                 errorCode = 8;
583             }
584
585             /*Print to the intermediate file and symbol file */
586             fprintf(intermediate_file, "%d\t\t\t%s\t%s\t%d\n", locctr,
                  tokenStructure.mnemonic, tokenStructure.opcode, errorCode);
587
588             /* Search for the mnemonic in the opcode table and add 3 to it  */
589             for (int i = 0; i < numMnemonics; i++)
590             {
591                 if (strcmp(opcodeStructure[i].mnemonic, tokenStructure.mnemonic)
                    == 0)
592                 {
593                     locctr += 3;
594                 }
595             }
596
597             /*Update the memory locations after LOCCTR is printed in the file
```

```
                */
598             if (strcmp(tokenStructure.mnemonic, "BYTE") == 0 || strcmp
                  (tokenStructure.mnemonic, "RESB") == 0 || strcmp
                  (tokenStructure.mnemonic, "RESW") == 0 || strcmp
                  (tokenStructure.mnemonic, "WORD") == 0)
599             {
600                 locctr += memLenght;
601             }
602         }
603         index++;
604     }
605
606     programLenght = locctr - start;
607     printf("Pass One complete successfully. \n");
608
609     fprintf(intermediate_file, "\n\n\t Printing Error Code List: \n\n");
610     fprintf(intermediate_file,
          "*================================================*\n");
611     fprintf(intermediate_file, "\tNo Error = 0\n");
612     fprintf(intermediate_file, "\tDuplicate Label = 1\n");
613     fprintf(intermediate_file, "\tIllegal Label = 2\n");
614     fprintf(intermediate_file, "\tIllegal Operation = 3\n");
615     fprintf(intermediate_file, "\tIllegal Data Storage Directive = 4\n");
616     fprintf(intermediate_file, "\tMissing START Directive = 5\n");
617     fprintf(intermediate_file, "\tMissing END Directive = 6\n");
618     fprintf(intermediate_file, "\tToo Many Symbols = 7\n");
619     fprintf(intermediate_file, "\tProgram Too Long = 8\n");
620     fprintf(intermediate_file,
          "*================================================*\n");
621
622     fclose(intermediate_file);
623     fclose(source_file);
624     fclose(symbol_file);
625 }
626 void passTwo()
627 {
628     printf("Pass Two is still in development. \n\n");
629 }
630
```