```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// implicit declaration of methods

void parseKeywords(int argc, char *argv[]);

void printKeywordOccurences(int count);

void walkthroughFile(int argc);

void tokenizeLine(char *line, int argc);

void checkIfTokenIsAKeyword(int argc, char *token);


/**

 * @brief

 * structure to store the keywords and their occurences

 */

struct keywordpair

{

  char *keyword;

  int count;

};


/**

 * @brief

 * declaration of the array of keywordpairs

 */

struct keywordpair *res;


/**

 * @brief
 *
 * @param argc number of command line arguments
 * @param argv array of command line arguments
 * @return int method return code
 */
int main(int argc, char *argv[])
{
  parseKeywords(argc, argv);

  // FILE *fp = readFile();
  // walkthroughFile(fp, argc);

  walkthroughFile(argc);

  printKeywordOccurences(argc);

  // fclose(fp);
  if (res)
    free(res);

  return 0;
}

/**
 * @brief
 * parses command line input and stores it in an array of keywordpairs which is declared globally
 */
void parseKeywords(int argc, char *argv[])
{
  // dynamic memory allocation based on the number of arguments passed
  // argc-1 because the first argument is the name of the program
  res = (struct keywordpair *)malloc((argc - 1) * sizeof(struct keywordpair));
  int i;
  for (i = 0; i < argc - 1; ++i)
  {
    (res + i)->keyword = argv[i + 1];
    // initialize the count of each keyword to 0
    (res + i)->count = 0;
  }
}

/**
 * @brief
 *
 * @param count keywords count
 */
void printKeywordOccurences(int count)
{
  printf("Displaying Information:\n");
  printf("Keyword\tOccurences\n");
  int i;
  for (i = 0; i < count - 1; ++i)
  {
    printf("%s\t%d\n", (res + i)->keyword, (res + i)->count);
  }
}
```

```c
// FILE *readFile()
// {
//   // FILE *fp = fopen("tintTale.txt", "r");
//   if (fp == NULL)
//     exit(EXIT_FAILURE);
//   return fp;
// }

/**
 * @brief
 *
 * @param argc arguments count
 * This will read the file(passed from command line via stdin) line by line and tokenize each
 * line and check if the token is a keyword
 *
 * at the end, it will free the memory allocated for the char pointer [line]
 */
void walkthroughFile(int argc)
{
  char *line = NULL;

  size_t len = 0;
  ssize_t read;
  long i = 0;

  while ((read = getline(&line, &len, stdin)) != -1)
  {
    tokenizeLine(line, argc);
  }
  if (line)
    free(line);
}

/**
 * @brief
 *
 * @param line char pointer to the line read from the file
 * @param argc arguments count
 *
 * This will tokenize the line and calls checkIfTokenIsAKeyword method
 *
 * stores the tokenized words in an array of char pointers
 * at the end, it will free the memory allocated for the char pointer [inputLine]
 */
void tokenizeLine(char *line, int argc)
{
  char *inputFile[2000];
  int i = 0;
  char *token = strtok(line, " ");
  while (token != NULL)
  {
    int len = strlen(token);
    inputFile[i] = malloc(sizeof(char) * len);
    strcpy(inputFile[i], token);
    i++;

    checkIfTokenIsAKeyword(argc, token);
    token = strtok(NULL, " ");
  }
  int l;
  for (l = 0; l < i; l++)
  {
    free(inputFile[l]);
  }
}

/**
 * @brief
 *
 * @param argc arguments count
 * @param token tokenized word
 *
 * check if the token is a keyword
 * if keyword is found, it will increment the count of the keyword
 *
 */
void checkIfTokenIsAKeyword(int argc, char *token)
{
  int i;
  for (i = 0; i < argc - 1; i++)
  {
    char *right = ((res + i)->keyword);
    int result = strcmp(token, right);
    if (result == 0)
    {
      (res + i)->count = (res + i)->count + 1;
      }
    }
  }
```