```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>

//Declaration of the function which reads all entries of a directory recursively
void readAllFilesOfThisDir(DIR *parentDir, int level, char *dir);

//stucture to store command line args.
struct cmd_args
{

  short hasPrintSizeFilter;
  short hasFileExtensionFilter;
  short hasMinSizeFilter;
  short hasTypeFilter;

  char fileType;
  long long minSize;
  char *extension;
};

// struct instance declaration
struct cmd_args argsConfig;

char *baseDir;

//Function to concatinate path and file names and returns complete file path.
```

```
char *getFileName(char *path, char *fileName)
{
  //Memory allocation
  char *n = (char *)malloc(1 + strlen(path) + strlen(fileName));
  strcpy(n, path);
  strcat(n, "/");
  strcat(n, fileName);

  return n;
}



//Returns file size of the files.
off_t fsize(const char *filename)
{
  struct stat st;

  if (lstat(filename, &st) == 0)
    return st.st_size;

  return 0;
}




//This the main function for this assignment.
int main(int argc, char **argv)
{
  // Flashes error if number of args is < 2
```

```
DIR *parentDir;
if (argc < 2)
{
  printf("Usage: %s <dirname>\n", argv[0]);
  exit(-1);
}
  int i;
for (i = 1; i < argc; i++)
{
  if (strcmp(argv[i], "-S") == 0)
  {

    argsConfig.hasPrintSizeFilter = 2;
    printf("has size argument\n");
  }
  if (strcmp(argv[i], "-s") == 0)
  {
    if (argv[i + 1] == NULL || strstr(argv[i + 1], "-"))
    {
      printf("After -s you should pass the max file size");
      exit(-1);
    }
    else
    {

      argsConfig.minSize = atoll(argv[i + 1]);
      argsConfig.hasMinSizeFilter = 2;
      printf("has size limit argument : %lld \n\n", argsConfig.minSize);
      i++;
    }
```

```
}

if (strcmp(argv[i], "-f") == 0)
{
  if (argv[i + 1] == NULL || strstr(argv[i + 1], "-"))
  {
    printf("After -f you should pass the file format");
    exit(-1);
  }
  else
  {
    argsConfig.extension = (char *)malloc(1 + strlen(argv[i + 1]));
    strcpy(argsConfig.extension, ".");
    strcat(argsConfig.extension, argv[i + 1]);
    argsConfig.hasFileExtensionFilter = 2;
    printf("has type argument: %s\n", argsConfig.extension);
    i++;
  }
}

if (strcmp(argv[i], "-t") == 0)
{
  if (argv[i + 1] == NULL || strstr(argv[i + 1], "-"))
  {
    printf("After -t you should pass either f or d");
    exit(-1);
  }
  else
  {
```

```
      argsConfig.hasTypeFilter = 2;
      argsConfig.fileType = argv[i + 1][0];
      printf("has type argument: %c\n", argsConfig.fileType);
      if (argsConfig.fileType != 'd' && argsConfig.fileType != 'f')
      {
        printf("After -t you should pass either f or d");
        exit(-1);
      }
      i++;
    }
  }
 }

 baseDir = argv[1];

 parentDir = opendir(baseDir);
 if (parentDir == NULL)
 {
   printf("Error opening directory '%s'\n", baseDir);
   exit(-1);
 }
 // int count = 1;

 readAllFilesOfThisDir(parentDir, 0, "");
 closedir(parentDir);
 return 0;
}

int printIfFormatMatches(char *nn)
 {
```

```
  if (argsConfig.hasFileExtensionFilter == 2)
  {
    char *end = strrchr(nn, '.');
    if (end && strcmp(end, argsConfig.extension) == 0)
      return 1;
    else
      return -1;
  }
  return 0;
}



//Function to print the filesize.
void printSizeOfFile(char *cdir, char *fname)
  {
  printf(" (%ld bytes)", fsize(getFileName(cdir, fname)));
  }

int count = 1;
// Function to print file name of file names matches.
void readAllFilesOfThisDir(DIR *parentDir, int level, char *currDir)
  {

  struct dirent *dirent;
  char cdir[1000] = "";
  int i;
  int printed = 0;
//  int shouldPrint = 0;

  while ((dirent = readdir(parentDir)) != NULL)
```

```
{
  printed = 0;
  if (strcmp(dirent->d_name, ".") != 0 && strcmp(dirent->d_name, "..") != 0)
  {

    // Condition for nested folders.
    if (strlen(currDir) > 1)
    {
      strcat(cdir, currDir);
    }
    else
    {
      strcpy(cdir, baseDir);
    }

    // checking for file format.
    int ffResult = printIfFormatMatches(dirent->d_name);
    long long fileSize = fsize(getFileName(cdir, dirent->d_name));

    // conditions for -s
    // for all files
    if (argsConfig.hasMinSizeFilter == 2 && DT_DIR != dirent->d_type)
    {
      if (fsize(getFileName(cdir, dirent->d_name)) > argsConfig.minSize)
      {

        if ((ffResult == 0 || ffResult == 1) &&
                          (argsConfig.hasTypeFilter == 0 ||
                           (argsConfig.hasTypeFilter == 2 &&
```

```
                        ((argsConfig.fileType == 'f' && DT_REG == dirent->d_type) || (argsConfig.fileType ==
'd' && DT_DIR == dirent->d_type)))))
        {

          printf("[%d]\t", count);
          for (i = 0; i < level; i++)
          {
            if (i % 2 == 0 || i == 0)
              printf("|");
            else
              printf(" ");
          }
          printf("|-%s", dirent->d_name);
          printf(" (%lld bytes)", fileSize);
          printf("\n");
          count++;
        }
      }
    }
    else
    {


      if (ffResult == 0 || ffResult == 1 || DT_DIR == dirent->d_type)
      {

        if (

          argsConfig.hasTypeFilter == 0 ||
          (argsConfig.hasTypeFilter == 2 &&
```

```
          ((argsConfig.fileType == 'f' && DT_REG == dirent->d_type) || (argsConfig.fileType == 'd' &&
DT_DIR == dirent->d_type))))
        {
          printf("[%d]\t", count);
          for (i = 0; i < level; i++)
          {
            if (i % 2 == 0 || i == 0)
              printf("|");
            else
              printf(" ");
          }
          if (argsConfig.hasPrintSizeFilter == 2)
          {
            printf("|-%s", dirent->d_name);
            printSizeOfFile(cdir, dirent->d_name);
            printed = 2;

          }
          else
          {

            printf("|-%s", dirent->d_name);
            printed = 2;
          }
        }
      }



      if (printed == 2)
```

```
    {
      printf("\n");
      count++;
    }

    strcat(cdir, "/");
    strcat(cdir, dirent->d_name);



    if (DT_DIR == dirent->d_type)
    {
      DIR *ddir = opendir(cdir);
      if (ddir != NULL)
      {

        readAllFilesOfThisDir(ddir, level + 2, cdir);
        closedir(ddir);
      }
    }
  }
}
else
{
  if (argsConfig.hasTypeFilter == 0 || (argsConfig.hasTypeFilter == 2 && argsConfig.fileType == 'd'))
  {
    // . ..
    printf("[%d]\t", count);
    for (i = 0; i < level; i++)
    {
      if (i % 2 == 0 || i == 0)
```

```
            printf("|");
          else
            printf(" ");
        }
        printf("|-%s\n", dirent->d_name);
        count++;
      }
    }
  }
}
```