```
search: main.c
   gcc -g -o search main.c
/*
Name: Atmuri Trinadh kumar
BlazerID: tkatmuri
project: HW02
To compile: make (search: main.c
   gcc -g -o search main.c)
To run: {
   ./search,
   ./search -S,
   ./search -s 1024 -f jpg 1
}
*/

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

#define MAX_PATH_SIZE 2000

char *array[8192];
int arrCount = 0;
int duplicateCount =0;

void sizeRestrictedPrint(struct dirent *dirent, int maxFileSize, int tabSpaces, int count, char
*path, char *flagStr, char *smallEcmd, int capitalE);
void stringPattern(struct dirent *dirent, int tabSpaces,char *pattern, int dirDepth, char *flagStr,
int maxFileSize, int count, char *path, char *smallEcmd, int capitalE);
void printReg(struct dirent *dirent, int tabSpaces, int count);
void printReg(struct dirent *dirent, int tabSpaces, int count);
void traverseDirectory(char *path, int tabSpaces, int maxFileSize, char *flagStr, char *pattern,
int dirDepth, int isReg, char *smallEcmd, char *capitalEcmd, int capitalE);
char *addStr(char *first, char *second);
void printExtraInfo(struct dirent *dirent, char *path, char *flagStr, char *smallEcmd);
void printOnly(struct dirent *dirent, int tabSpaces, char *path, char *type);
void smallEfunc(struct dirent *dirent, char *smallEcmd, char *path);
void capitalEfunc();
```

```
char *filetype(unsigned char type) {
  char *str;
  switch(type) {
  case DT_BLK: str = "block device"; break;
  case DT_CHR: str = "character device"; break;
  case DT_DIR: str = "directory"; break;
  case DT_FIFO: str = "named pipe (FIFO)"; break;
  case DT_LNK: str = "symbolic link"; break;
  case DT_REG: str = "regular file"; break;
  case DT_SOCK: str = "UNIX domain socket"; break;
  case DT_UNKNOWN: str = "unknown file type"; break;
  default: str = "UNKNOWN";
  }
  return str;
}

void printOnly(struct dirent *dirent, int tabSpaces, char *path,char *type) {

    char *fType = filetype(dirent->d_type);

    if(strcmp(fType, "regular file") == 0 && strcmp(type,"p") == 0 ) {
       printf("%*s %s (%s)\n",4*tabSpaces," ", dirent->d_name, filetype(dirent->d_type));
    }
    if(strcmp(fType, "directory") == 0 && strcmp(type, "d") == 0) {
       printf("%*s %s (%s)\n",4*tabSpaces," ", dirent->d_name, filetype(dirent->d_type));
    }
}

void printReg(struct dirent *dirent, int tabSpaces, int count) {
    printf("%*s[%d] %s (%s)\n", 4 * tabSpaces, " ", count, dirent->d_name, filetype(dirent-
>d_type));

}

void printExtraInfo(struct dirent *dirent, char *path, char *flagStr, char *smallEcmd) {

    struct stat buf;
    char *x = malloc(500);
    strcpy(x,path);
    strcat(x,"/");
    strcat(x,dirent->d_name);
    stat(x, &buf);
```

```
printf("%s", dirent->d_name);
off_t fByteSize = buf.st_size;
if(S_ISDIR(buf.st_mode)) {
   printf("\t0 ");
}
else {
   printf("\t%lld ",fByteSize);
}

if (S_ISDIR(buf.st_mode))
   putchar('d');
else
   putchar('-');
if ((buf.st_mode & S_IRUSR) != 0)
   putchar('r');
else
   putchar('-');
if ((buf.st_mode & S_IWUSR) != 0)
   putchar('w');
else
   putchar('-');
if ((buf.st_mode & S_IXUSR) != 0)
   putchar('x');
else
   putchar('-');
if ((buf.st_mode & S_IRGRP) != 0)
   putchar('r');
else
   putchar('-');
if ((buf.st_mode & S_IWGRP) != 0)
   putchar('w');
else
   putchar('-');
if ((buf.st_mode & S_IXGRP) != 0)
   putchar('x');
else
   putchar('-');
if ((buf.st_mode & S_IROTH) != 0)
   putchar('r');
else
   putchar('-');
if ((buf.st_mode & S_IWOTH) != 0)
```

```
        putchar('w');
    else
        putchar('-');
    if ((buf.st_mode & S_IXOTH) != 0)
        putchar('x');
    else
        putchar('-');

    printf("\t%s", ctime(&buf.st_atime));

}

void smallEfunc(struct dirent *dirent, char *smallEcmd, char *path) {

    char *fType = filetype(dirent->d_type);

    array[arrCount] = path;
    arrCount = 1+arrCount;
    array[arrCount] = (char* )NULL;

    if(strcmp(fType, "directory") != 0) {
        pid_t pid = fork();
        if(pid < 0) {
            printf("Error in forking\n");
            exit(-1);
        }
        else if(pid == 0) {
            execvp(array[0], array);
            printf("Error in execution\n");
            exit(-1);
        }
        else {
            waitpid(pid, NULL, 0);
            printf("\n");
        }
    }

    arrCount = duplicateCount;
}

void capitalEfunc() {

    array[arrCount] = (char *)NULL;
```

```
        pid_t pid = fork();
        if(pid < 0) {
            printf("Error in forking\n");
            exit(-1);
        }
        else if(pid == 0) {

            execvp(array[0], array);
            printf("Error in execution\n");
            exit(-1);
        }
        else {
            waitpid(pid, NULL, 0);
            printf("\n");
        }
}

void sizeRestrictedPrint(struct dirent *dirent, int maxFileSize, int tabSpaces, int count, char
*path, char *flagStr, char *smallEcmd, int capitalE) {

    struct stat buf;
    char *x = malloc(5000);
    strcpy(x,path);
    strcat(x,"/");
    strcat(x,dirent->d_name);
    stat(x, &buf);

    off_t fByteSize = buf.st_size;
    if(fByteSize < maxFileSize && strstr(flagStr, "S") != NULL) {
        printExtraInfo(dirent, x, flagStr, smallEcmd);
    }
    else if(fByteSize < maxFileSize && strstr(flagStr, "e") != NULL) {
        smallEfunc(dirent, smallEcmd, x);
    }
    else if(fByteSize < maxFileSize && capitalE == 1) {
        if(strstr(x, "git") != NULL) {
            printReg(dirent, tabSpaces, count);
        }
        else{
            array[arrCount] = x;
            arrCount = 1+arrCount;
            printReg(dirent, tabSpaces, count);
```

```
        }

    }
    else if(dirent->d_type == DT_DIR || fByteSize < maxFileSize) {
        printf("%*s[%d] %s \n", 4 * tabSpaces, " ", count, dirent->d_name);
    }

}

void stringPattern(struct dirent *dirent, int tabSpaces,char *pattern, int dirDepth, char *flagStr,
int maxFileSize, int count, char *path, char *smallEcmd, int capitalE) {

    char *x = malloc(500);
    strcpy(x,path);
    strcat(x,"/");
    strcat(x,dirent->d_name);

    if (strstr(flagStr, "s") != NULL && tabSpaces <= dirDepth && strstr(dirent->d_name, pattern)
!= NULL) {
        sizeRestrictedPrint(dirent, maxFileSize, tabSpaces, count, path, flagStr, smallEcmd,
capitalE);
    }
    else if(strstr(flagStr, "S") != NULL && tabSpaces <= dirDepth && strstr(dirent->d_name,
pattern) != NULL) {
        printExtraInfo(dirent, path, flagStr, smallEcmd);
    }
    else if(strstr(flagStr, "e") != NULL && tabSpaces <= dirDepth && strstr(dirent->d_name,
pattern) != NULL) {
        smallEfunc(dirent, smallEcmd, x);
    }
    else if(strstr(dirent->d_name, pattern) != NULL && tabSpaces < dirDepth && capitalE == 1 &&
strstr(flagStr, "s") != NULL) {
        array[arrCount] = x;
        arrCount = 1+arrCount;
        sizeRestrictedPrint(dirent, maxFileSize, tabSpaces, count, path, flagStr, smallEcmd,
capitalE);
    }
    else if(strstr(dirent->d_name, pattern) != NULL && tabSpaces < dirDepth && capitalE == 1) {
        if(strstr(x, "git") != NULL) {
            printReg(dirent, tabSpaces, count);
        }
        else{
            array[arrCount] = x;
```

```
            arrCount = 1+arrCount;
            printReg(dirent, tabSpaces, count);
        }
    }
    else if(strstr(dirent->d_name, pattern) != NULL && tabSpaces<=dirDepth) {
        printf("%*s %s (%s)\n", 4 * tabSpaces, " ", dirent->d_name, filetype(dirent->d_type));
    }
}

void traverseDirectory(char *path, int tabSpaces, int maxFileSize, char *flagStr, char *pattern,
int dirDepth, int isReg, char *smallEcmd, char *capitalEcmd, int capitalE) {
    struct dirent *dirent;
    DIR *parentDir;
    // First, we need to open the directory.
    parentDir = opendir(path);
    if (parentDir == NULL) {
        printf ("Error opening directory '%s'\n", path);
        exit (-1);
    }
    int count = 1;
    // After we open the directory, we can read the contents of the directory, file by file.
    while((dirent = readdir(parentDir)) != NULL){
        // If the file's name is "." or "..", ignore them. We do not want to  xinfinitely recurse.
        if (strcmp(dirent->d_name, ".") == 0 || strcmp(dirent->d_name, "..") == 0)
        {
            continue;
        }

        // all flags
        // f, s & S
        if(strstr(flagStr, "f") != NULL && strstr(flagStr, "s") != NULL && strstr(flagStr, "S") != NULL
&& strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
            stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
        }
        //three flags
        // s f and e
        else if(strstr(flagStr, "s") != NULL && strstr(flagStr, "e") != NULL && strstr(flagStr,"S") ==
NULL && strstr(flagStr, "f") != NULL && strstr(flagStr, "E") == NULL) {
            stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
        } // s f and E
```

```
    else if(strstr(flagStr, "s") != NULL && strstr(flagStr, "E") != NULL && strstr(flagStr,"S") ==
NULL && strstr(flagStr, "f") != NULL && strstr(flagStr, "e") == NULL) {
        stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
    }

    // double flags
    // s and f
    else if (strstr(flagStr, "s") != NULL && strstr(flagStr, "f") != NULL && strstr(flagStr, "S") ==
NULL && strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
        stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
    }
    // s and S
    else if(strstr(flagStr, "s") != NULL && strstr(flagStr, "S") != NULL && strstr(flagStr, "f") ==
NULL && strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
        sizeRestrictedPrint(dirent, maxFileSize, tabSpaces, count, path, flagStr, smallEcmd,
capitalE);
    }
    // S and f
    else if(strstr(flagStr, "S") != NULL && strstr(flagStr, "f") != NULL && strstr(flagStr, "s") ==
NULL && strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
        stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
    }
    // s and e
    else if(strstr(flagStr, "s") != NULL && strstr(flagStr, "e") != NULL && strstr(flagStr,"S") ==
NULL && strstr(flagStr, "f") == NULL && strstr(flagStr, "E") == NULL) {
        sizeRestrictedPrint(dirent, maxFileSize, tabSpaces, count, path, flagStr, smallEcmd,
capitalE);
    } // f and e
    else if (strstr(flagStr, "f") != NULL && strstr(flagStr, "e") != NULL && strstr(flagStr,"S") ==
NULL && strstr(flagStr, "s") == NULL && strstr(flagStr, "E") == NULL) {
        stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
    } // s and E
    else if (strstr(flagStr, "s") != NULL && strstr(flagStr, "E") != NULL && strstr(flagStr,"S") ==
NULL && strstr(flagStr, "f") == NULL && strstr(flagStr, "e") == NULL) {
        if(dirent->d_type != DT_DIR) {
            sizeRestrictedPrint(dirent, maxFileSize, tabSpaces, count, path, flagStr, smallEcmd,
capitalE);
        }
    } // f and E
```

```
    else if (strstr(flagStr, "f") != NULL && strstr(flagStr, "E") != NULL && strstr(flagStr,"S") ==
NULL && strstr(flagStr, "s") == NULL && strstr(flagStr, "e") == NULL) {
        if(dirent->d_type != DT_DIR) {
          stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
        }
    }


    // single flags
    // s
    else if (strstr(flagStr, "s") != NULL && strstr(flagStr, "f") == NULL && strstr(flagStr, "S") ==
NULL && strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
        sizeRestrictedPrint(dirent, maxFileSize, tabSpaces, count, path, flagStr, smallEcmd,
capitalE);
    }
    // f
    else if(strstr(flagStr, "f") != NULL && strstr(flagStr, "s") == NULL && strstr(flagStr, "S") ==
NULL && strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
        stringPattern(dirent, tabSpaces, pattern, dirDepth, flagStr, maxFileSize, count, path,
smallEcmd, capitalE);
    }
    // S
    else if(strstr(flagStr, "S") != NULL && strstr(flagStr, "f") == NULL && strstr(flagStr, "s") ==
NULL && strstr(flagStr, "e") == NULL && strstr(flagStr, "E") == NULL) {
        printExtraInfo(dirent, path, flagStr, smallEcmd);
    }
    // e
    else  if(strstr(flagStr, "e") != NULL && strstr(flagStr, "S") == NULL && strstr(flagStr, "f") ==
NULL && strstr(flagStr, "s") == NULL && strstr(flagStr, "E") == NULL) {
        // dirent

        if(dirent->d_type != DT_DIR) {
          char *x = malloc(500);
          strcpy(x,path);
          strcat(x,"/");
          strcat(x,dirent->d_name);
          smallEfunc(dirent, smallEcmd, x);
        }


    }
    // Regular Print
```

```
      else if(isReg != 0) {
         printReg(dirent, tabSpaces, count);
      }

      if(strstr(flagStr, "p") != NULL) {
         printOnly(dirent, tabSpaces, path, "p");
      } else if(strstr(flagStr, "d") != NULL) {
         printOnly(dirent, tabSpaces, path, "d");
      }

      char *z = (char *) malloc(MAX_PATH_SIZE);
      strcpy(z, path);
      strcat(z, "/");
      strcat(z, dirent->d_name);

      if(capitalE == 1 && dirent->d_type != DT_DIR && strstr(flagStr, "s") == NULL &&
strstr(flagStr, "f") == NULL) {
            array[arrCount] = z;
            arrCount = 1+arrCount;
      }

      // Check to see if the file type is a directory. If it is, recursively call traverseDirectory on it.
      if (dirent->d_type == DT_DIR) {
         // Build the new file path.
         char *subDirPath = (char *) malloc(MAX_PATH_SIZE);
         strcpy(subDirPath, path);
         strcat(subDirPath, "/");
         strcat(subDirPath, dirent->d_name);
         traverseDirectory(subDirPath, tabSpaces + 1, maxFileSize, flagStr, pattern, dirDepth,
isReg, smallEcmd, capitalEcmd, capitalE);
      }
   }
}

void decryptArguments(int argc, char *argv[], char **pattern, char **path, int *dirDepth, int
*maxFileSize, int *onlyFiles, int *extraNeeded, int *sizeLimitation, int *depthNeeded, int
*smallE, int *capitalE, char **smallEcmd, char **capitalEcmd) {
   int gotPath = 0;
   for (int i=1; i<argc; i++) {
      if(argv[i][0] == '-') {
         if(argv[i][1] == 'S'){
            *extraNeeded = 1;
         }
```

```
            else if(argv[i][1] == 's'){
               *maxFileSize = atoi(argv[i+1]);
               *sizeLimitation = 1;
               i++;
            }
            else if(argv[i][1] == 'f') {
               *pattern = argv[i+1];
               *dirDepth = atoi(argv[i+2]);
               *depthNeeded = 1;
               i = i+2;
            }
            else if(argv[i][1] == 't') {
               if(strcmp("f", argv[i+1]) == 0) {
                  *onlyFiles = 1;
                  i++;
               }
               else {
                  *onlyFiles = 0;
                  i++;
               }
            }
            else if(argv[i][1] == 'e') {
               *smallE = 1;
               *smallEcmd = argv[i+1];
               i++;
            }
            else if(argv[i][1] == 'E') {
               *capitalE = 1;
               *capitalEcmd = argv[i+1];
               i++;
            }
         }
         else {
            *path = argv[i];
            gotPath = 1;
         }
      }
      if (gotPath == 0) {
            *path = "./";
         }
}

int main(int argc, char *argv[]) {
```

```
   int tabSpaces = 0;

   char *pattern, *smallEcmd, *capitalEcmd, *flagStr = malloc(50);
   char *path = NULL;
   int dirDepth = -1, maxFileSize = -1, onlyFiles = -1, extraNeeded = 0, sizeLimitation = 0,
depthNeeded = 0, isReg = 1, smallE = 0, capitalE = 0;
   strcpy(flagStr, "");

   decryptArguments(argc, argv, &pattern, &path, &dirDepth, &maxFileSize, &onlyFiles,
&extraNeeded, &sizeLimitation, &depthNeeded, &smallE, &capitalE, &smallEcmd,
&capitalEcmd);

   if(extraNeeded == 1) {
      strcat(flagStr, "S");
   }
   if(sizeLimitation == 1) {
      strcat(flagStr, "s");
   }
   if(depthNeeded == 1) {
      strcat(flagStr, "f");
   }
   if(onlyFiles == 1) {
      strcat(flagStr, "p");
   } else if(onlyFiles == 0) {
      strcat(flagStr, "d");
   }
   if(smallE == 1) {
      strcat(flagStr, "e");
      char *token = strtok(smallEcmd, " ");

      while(token != NULL) {
         array[arrCount] = token;
         token = strtok(NULL, " ");
         arrCount= 1+arrCount;

   }
   }
   if(capitalE == 1) {
      strcat(flagStr, "E");
      char *token = strtok(capitalEcmd, " ");

      while(token != NULL) {
         array[arrCount] = token;
```

```
            token = strtok(NULL, " ");
            arrCount= 1+arrCount;
        }
        }

        duplicateCount = arrCount;

        if (argc < 2) {
            traverseDirectory(path ,tabSpaces, maxFileSize, flagStr, pattern, dirDepth, isReg,
smallEcmd, capitalEcmd, capitalE);
        }
        else if(onlyFiles == -1) {
            traverseDirectory(path ,tabSpaces, maxFileSize, flagStr, pattern, dirDepth, isReg,
smallEcmd, capitalEcmd, capitalE);
        }
        else {
            isReg = 0;
            traverseDirectory(path ,tabSpaces, maxFileSize, flagStr, pattern, dirDepth, isReg,
smallEcmd, capitalEcmd, capitalE);
        }
        if (capitalE == 1) {
            capitalEfunc();
        }
        return 0;
}
```