```
#!/bin/bash

error()  {
        echo $1 >&2
        cleanup
        exit $2
}

# prepare
#       prepares our compelation process
prepare() {
        if [ ! -f $1 ]; then
                error "File $1 does not exist" -1
        fi
    clean=`echo $1 | sed 's/\.\///'`
        tmp=`mktemp -dt ${clean}XXXXX`
}

# interpret
#       runs the compiler on the .c file and executes it.
interpret() {
        execname=$1
        filename=$2
        scriptloc=$3
        tmpdir=$4
        shift 4
        /usr/bin/gcc -o "$tmpdir/$execname" "$scriptloc/$filename";
        if [[ $? -eq 0 ]]; then
                $tmpdir/$execname "$@"
        else
                error "Error in gcc compiler" -3
        fi
}

# cleanup
#       cleanup the resulting files in the tmp directory
cleanup() {
        /bin/rm -rf $tmp
}

trap '{cleanup; exit 1}' SIGHUP SIGINT SIGQUIT SIGTERM
cprog="$0.c"
prepare $cprog
interpret $0 $cprog $PWD $tmp "$@"
cleanup
================================================================================
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdarg.h>

// params
```

```
//	container class for the parameters of the program.
//	There is a segment, count and file. Each file is
//	read count lines for each segment.
typedef char* string;
typedef struct params_s {
	int segment;
	int count;
	string* files;
} params_t;

params_t params_ctor(int N, int M, string files) {
	params_t t;
	t.segment = N;
	t.count = M;
	t.files = &files;
	return t;
}

// use getopts to get M,N and the file name.
// for the running programming. If there is no
// m and n argument
params_t get_parameters(int argc, string* argv) {
	char c;
	int M = 0, N = 0, use_env = 1, buffer = 0, *cur;
	cur = &N;
	while ((c = getopt(argc, argv, "1234567890,")) != -1) {
		use_env = 0;
		switch (c) {
			case '1': case '2': case '3': case '4': case'5':
			case '6': case '7': case '8': case '9': case'0':
				*cur = (*cur * 10) + (c - '0');
				break;

			case ',':
				if (cur != &M && cur != &buffer) {
					cur = &M;
				} else if (cur != &buffer) {
					cur = &buffer;
				}
				break;
		}
	}
	return params_ctor(N, M, NULL);
}

// copy_string_array
//	utility method to copy string arrays.
void copy_string_array(const int copy_length, string* dest, const string* src) {
	int i;
	for (i = 0; i < copy_length; i++) {
		dest[i] = malloc(sizeof(char) * strlen(src[i]));
		strcpy(dest[i], src[i]);
	}
}
```

```
void copy_string_arguments(const int copy_length, string* dest, ...) {
        int i;
        va_list vl;
        va_start(vl, dest);
        for (i = 0; i < copy_length; i++) {
                string elem = va_arg(vl, char*);
                dest[i] = malloc(sizeof(char) * strlen(elem));
                strcpy(dest[i], elem);
        }
        va_end(vl);
}

int get_file_names(int argc, const string *argv, string *files) {
    int i, count = 0;
    for (i = 1; i < argc; i++) {
        if (argv[i][0] != '-') {
            files[count] = malloc(sizeof(char) * strlen(argv[i]));
            strcpy(files[count++], argv[i]);
        }
    }
    return count;
}

void read_file(FILE *f, const int N, const int M) {
        string line = malloc(sizeof(char) * BUFSIZ);
        int in_segment = 0, i = 0, j = 0;
        while (NULL != fgets(line, BUFSIZ, f)) {
                if (!(i % N) && !in_segment) {
                        in_segment = 1;
                }

                if (in_segment) {
                        if (j < M) {
                                printf("%s", line);
                                j++;
                        } else {
                                in_segment = 0;
                                j = 0;
                        }
                }
                i++;
        }
}

int main(int argc, string* argv) {
        // check too see what argument we need to use
    int number_of_files = 0;
        params_t args;
        string envarg = "", files[argc];
        if (argc >= 2 && argv[1][0] == '-') {
                args = get_parameters(argc, argv);
        } else if ((envarg = getenv("EVERY")) != NULL) {
                string pargs[2];
                copy_string_arguments(2, pargs, argv[0], envarg);
                args = get_parameters(2, pargs);
        } else {
                args.segment = 1;
                args.count = 1;
        }
    number_of_files = get_file_names(argc, argv, files);
    if (!number_of_files) {
        // read from stdin
                read_file(stdin, args.segment, args.count);
    } else {
                int i;
                for (i = 0; i < number_of_files; i++) {
                        read_file(fopen(files[i], "r"), args.segment, args.count);
                }
    }
        return 0;
}
================================================================================
// Ian schweer
// 22514022
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <locale.h>
#include <langinfo.h>
#include <errno.h>
#include <stdint.h>

#define ALMOST_ALL 1
#define ALL 2
typedef char* string;
typedef struct opts_s {
    int show_all;
    int deref_links;
    string *names;
    int numfiles;
} opt_t;

typedef struct statpod_s {
    struct stat s; // stat structure returned from stat
    struct stat l; // stat structure returned from lstat;
} statpod_t;

typedef struct node_s {
    string name;
    struct dirent* dp;
    struct stat statbuf;
    statpod_t *pod;
} node_t;

string readlink_name(string name);
int compnode(const void *a, const void *b);
void iteratenodes(node_t *data, int length);
statpod_t *statdir(string name, opt_t options);
node_t *getfiles(const char *path, node_t *files, int *j, opt_t options);
opt_t *getoptions(int argc, string* argv);
string getfilepath(char *name, const char *path);

int main(int argc, string* argv) {
    int i = 0, size = 0;
    opt_t *options;
    options = getoptions(argc, argv);
    for (i = 0; i < options->numfiles; i++) {
        node_t nodes[BUFSIZ];
        // get the files stat pod
        statpod_t *pod = statdir(options->names[i], *options);
        if (pod != NULL) {
            struct stat x = options->deref_links ? pod->s : pod->l;
            if (S_ISDIR(x.st_mode)) {
                // if the file is a directory, then go through this method
                if (options->numfiles > 1) printf("%s\n", options->names[i]);
                getfiles(options->names[i], nodes, &size, *options);
                qsort(nodes, size, sizeof(node_t), compnode);
                iteratenodes(nodes, size);
            } else {
                // the file is normal, print
                nodes[0].name = options->names[i];
                nodes[0].statbuf = options->deref_links ? pod->s : pod->l;
                iteratenodes(nodes, 1);
            }
            free(pod);
            free(options->names[i]);
        } else {
            perror("Unable to stat argument");
        }
    }
    free(options->names);
    return 0;
}

opt_t *getoptions(int argc, string* argv) {
    char c;
    int numflags = 0;
    opt_t *options = calloc(1, sizeof(opt_t));
    while ((c = getopt(argc, argv, "LaA")) != -1) {
        numflags++;
        switch (c) {
            case 'L':
                options->deref_links = 1;
                break;
            case 'a':
                options->show_all = ALL;
                break;
            case 'A':
                options->show_all = ALMOST_ALL;
                break;
        }
    }

    // get all the file names.
    options->names = calloc(argc, sizeof(char*));
    // default options[0].
    options->names[0] = (char*)calloc(1, sizeof(char)*2);
    strcpy(options->names[0] , ".");
    int i, j = 0;
    for (i = optind; i < argc; i++) {
       if (argv[i][0] != '-') {
            options->names[j] = calloc(1, sizeof(char) * strlen(argv[i]));
            strcpy(options->names[j++], argv[i]);
       }
    }
    options->numfiles = j ? j : 1;
    return options;
}

node_t *getfiles(const char *path, node_t *files, int *j, opt_t options) {
    struct dirent *dp;
    struct stat statbuf;
    DIR *_dir = opendir(path);
    int i = 0, dircount = 0, k = 0;
    char *dirs[BUFSIZ];
    long pathsize = 0;
    // Get all the files.
    while ((dp = readdir(_dir)) != NULL) {
       if (options.show_all == ALMOST_ALL && (!strcmp(dp->d_name, ".") || !strcmp(dp->d_name, ".."))) continue;
       if (!options.show_all && dp->d_name[0] == '.') continue;
       char *fullname = getfilepath(dp->d_name, path);
       statpod_t *statpod_ptr = statdir(fullname, options);
       if (statpod_ptr != NULL) {
           memcpy(&(files[i].statbuf), (options.deref_links ? &statpod_ptr->s : &statpod_ptr->l), sizeof(struct stat));
           files[i].name = calloc(1,sizeof(char) * strlen(dp->d_name));
           memcpy(files[i].name, dp->d_name, strlen(dp->d_name));
           files[i].name[strlen(dp->d_name)] = '\0';
           files[i++].dp = dp;
       } else {
            perror("Error using stat return value");
       }
       free(statpod_ptr);
       free(fullname);
    }
    *j = i;
    closedir(_dir);
}
```

```
int compnode(const void *a, const void *b) {
   int diff = (*(node_t*)a).statbuf.st_size - (*(node_t*)b).statbuf.st_size;
   if (!diff) return 0;
   else if (diff > 0) return -1;
   else return 1;
}

void iteratenodes(node_t *data, int length) {
      int i;
      struct passwd *pwd;
      struct group *grp;
      struct tm *tm, *curr;
      char datestring[256];
      int modecount = 9;
      mode_t modes[] = {S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH,
S_IXOTH};
      char outs[] = {'r', 'w', 'x'};

      // get the current time.
      time_t rawtime;
      time(&rawtime);
      for (i = 0; i < length; i++) {
      string temp_stuff;
      if (S_ISLNK(data[i].statbuf.st_mode)) {
         string linkname = readlink_name(data[i].name);
         temp_stuff = calloc(1, sizeof(char) * (strlen(data[i].name) + strlen(linkname) + strlen(" -> ")));
         sprintf(temp_stuff, "%s -> %s", data[i].name, linkname);
      } else {
         temp_stuff = calloc(1, sizeof(char) * strlen(data[i].name));
         strcpy(temp_stuff, data[i].name);
      }
            if (S_ISDIR(data[i].statbuf.st_mode))
                  putchar('d');
            else
                  putchar('-');

            int j = 0;
            for (j = 0; j < modecount; j++)
                  if (data[i].statbuf.st_mode & modes[j])
                        putchar(outs[j % 3]);
                  else
                        putchar('-');
            printf("%2d", data[i].statbuf.st_nlink);

            if ((pwd = getpwuid(data[i].statbuf.st_uid)) != NULL) {
         printf(" %-8.8s", pwd->pw_name);
      }
            else {
         printf(" %-8d", data[i].statbuf.st_uid);
      }



            if ((grp = getgrgid(data[i].statbuf.st_gid)) != NULL)
```

```
                        printf(" %-6.8s", grp->gr_name);
                else
                        printf(" %-8d", data[i].statbuf.st_gid);

                printf("%5jd ", data[i].statbuf.st_size);
                tm = localtime(&data[i].statbuf.st_mtime);

                if (difftime(rawtime, mktime(tm)) < 15768000) {
                        strftime(datestring, sizeof(datestring), "%b %e %R", tm);
                        printf("%s %s\n", datestring, temp_stuff);
                } else {
                        strftime(datestring, sizeof(datestring), "%b %e %Y", tm);
                        printf("%s %s\n", datestring, temp_stuff);
                }
        }
}

statpod_t *statdir(string name, opt_t options) {
    statpod_t *pod = calloc(1,sizeof(statpod_t));
    int lstat_return = !options.deref_links ? lstat(name, &(pod->l)) : 0;
    int stat_return = options.deref_links ? stat(name, &(pod->s)) : 0;

    if (lstat_return == -1) {
        char buf[100];
        sprintf(buf, "Error while stating file %s", name);
        perror(buf);
        return NULL;
    }
    if (lstat_return != -1 && stat_return == -1) {
        // the symbolic link points to nothing.
        perror(readlink_name(name));
        return NULL;
    }
    return pod;
}

string readlink_name(string name) {
    string buf = calloc(1, sizeof(char) * 256);
    if (readlink(name, buf, 256) == -1) {
        sprintf(buf, "reading link %s", buf);
        perror(buf);
    }
    return buf;
}

string getfilepath(char *name, const char *path) {
    string fullname = calloc(1, sizeof(char) * (strlen(name) + strlen(path) + 2));
    strcpy(fullname, path);
    strcat(fullname, "/");
    strcat(fullname, name);
    fullname[strlen(name) + strlen(path) + 1] = '\0';
    return fullname;
}
```