北京郵電大學

实验报告

题目：遍历文件目录

学　　院　：计算机学院

班　　级　：2018211302

姓　　名　：胡天翼

学　　号　：2018210547

指导教师　：蒋砚军

2021 年 4 月 24 日

编程作业：遍历文件目录

一、实验题目

编程实现程序list.c，列表普通磁盘文件，包括文件名和文件大小。

二、实验要求

对选项的处理，自行编程逐个分析命令行参数。不考虑多选项挤在一个命令行参数内的情况：

- 与ls命令类似，处理对象可以有0到多个
  - 0个：列出当前目录下所有文件
  - 普通文件：列出文件
  - 目录：列出目录下所有文件
- 实现自定义选项r,a,l,h,m以及--
  - r 递归方式列出子目录（每项要含路径，类似find的-print输出风格，需要设计递归程序）
  - a 列出文件名第一个字符为圆点的普通文件（默认情况下不列出文件名首字符为圆点的文件）
  - l 后跟一整数，限定文件大小的最小值（字节）
  - h 后跟一整数，限定文件大小的最大值（字节）
  - m 后跟一整数n，限定文件的最近修改时间必须在n天内
  - -- 显式地终止命令选项分析

以Linux提示语句风格表示该要求：

```
Usage: list1 [OPTION]... [FILE]...
List information about the FILEs (the current directory by default)

- a           Do not hide entries starting with .
- r           List subdirectories recursively
- l <bytes>   Minimum of file size
- h <bytes>   Maximum of file size
- m <days>    Limit file last modified time
```

三、实验目的

- 使用vi编辑文件，熟悉工具vi。
- 使用Linux的系统调用和库函数。
- 体会Shell文件通配符的处理方式以及命令对选项的处理方式。

四、实验原理

4.1 文件系统的存储结构

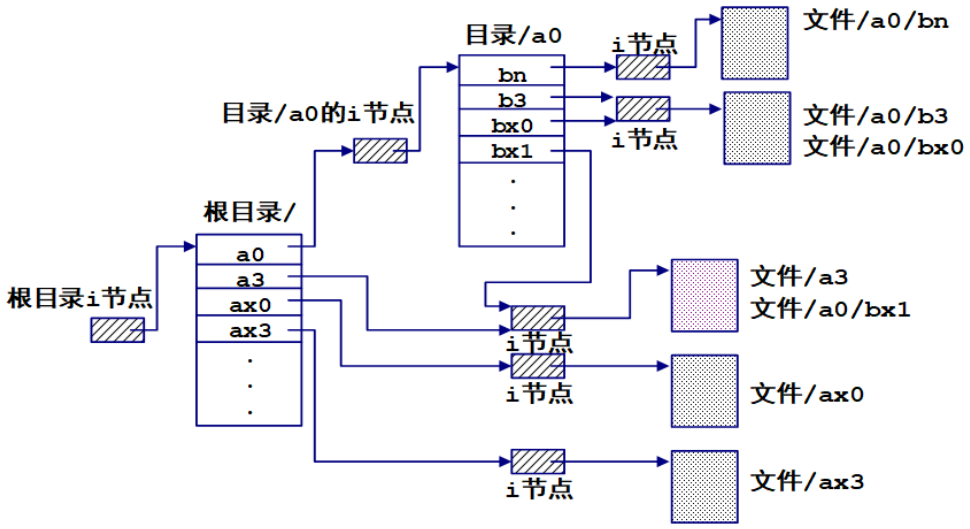
文件系统存储结构如下图所示：

引导块	专用块	i 节 点 区	文 件 存 储 区

其中，i节点区为指向文件存储区数据块的一些索引（index）指针（组成文件的逻辑块与硬盘的物理块之间的映射），存有文件的文件类型，属主，组，权限，link数，大小，时戳，但不含文件名。

文件存储区用于存放文件数据的区域，包括目录表。每个目录表也作为一个文件来管理，存于“文件存储区”中，有其自己的i节点和数据存储块。目录表由若干个“目录项”构成，目录项只含两部分信息：文件名与i节点号。

目录和i节点的存储结构如下图所示：



因此，获取当前目录下普通磁盘文件信息的方式为：获取文件名→获取文件i节点

获取当前目录下目录文件信息的方式为：获取目录名→获取文件i节点→读取目录表项→读取文件名与对应i节点号

4.2 i节点和目录数据结构

通过函数stat得到指定路径名的文件的i节点，函数fstat得到已打开文件的i节点

stat和fstat将数据放入调用者提供的stat结构中

函数原型：

```
int stat(const char *pathname, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

stat结构包含以下数据：

```
struct stat {
   dev_t     st_dev;         /* 存储该文件的块设备的设备号ID */
   ino_t     st_ino;         /* inode号 */
   mode_t    st_mode;        /* 访问权限及文件类型 */
   nlink_t   st_nlink;       /* link数 */
   uid_t     st_uid;         /* 文件主ID */
   gid_t     st_gid;         /* 组ID */
   dev_t     st_rdev;        /* device ID (if special file) */
   off_t     st_size;        /* 文件大小（字节数）*/
   blksize_t st_blksize;     /* blocksize for filesystem I/O */
   blkcnt_t  st_blocks;      /* 分配的512字节尺寸块个数 */
   struct timespec st_atim;  /* access时间 */
   struct timespec st_mtim;  /* modification时间 */
   struct timespec st_ctim;  /* change时间 */
};
```

st_dev

```
st_dev：存储该文件的块设备的设备号，包括主设备号与次设备号
例如：stat命令显示文件Device: 821h/2081d
十六进制0821，主设备号8（高字节），次设备号33(低字节)，  /dev/sdc1
ls -l /dev | grep '^b.* 8, *33'
brw-rw---- 1 root disk      8,  33 Nov 18 10:40 sdc1
```

st_mode

```
文件的基本存取权限和SUID/SGID权限(11比特)以及文件的类型(若干比特)
文件类型判st_mode & S_IFMT
S_IFREG      普通磁盘文件
S_IFDIR      目录文件
S_IFCHR      字符设备文件
S_IFIFO       管道文件
S_IFLNK      符号连接文件
```

st_size与st_blocks

```
程序可以通过st_size获取文件大小。
一般情况:st_size ≤ st_blocks * 512
稀疏文件:st_size > st_blocks * 512
```

st_ctim，st_atim，st_mtim域

```
Linux中存储这三个时间的精度为纳秒
“a访问”：读，执行（有些系统为了效率做懒惰处理，不更新，但不早于m时间）
“m修改”：文件内容修改。写文件
“c改变”：i节点信息变化。写文件，修改权限/link数/文件主等（m变，c也变）
```

4.3 访问目录

目录访问的一组库函数

```
#include <dirent.h>
DIR *opendir(char *dirname);
struct dirent *readdir(DIR *dir);
int closedir(DIR *dir);
```

dirent结构体：

```
struct dirent
{
    long d_ino; /* inode number 索引节点号 */
     off_t d_off; /* offset to this dirent 在目录文件中的偏移 */
     unsigned short d_reclen; /* length of this d_name 文件名长 */
     unsigned char d_type; /* the type of d_name 文件类型 */
     char d_name [NAME_MAX+1]; /* file name (null-terminated) 文件名，最长255字符 */
}
```

opendir打开目录得到句柄（NULL表示失败）

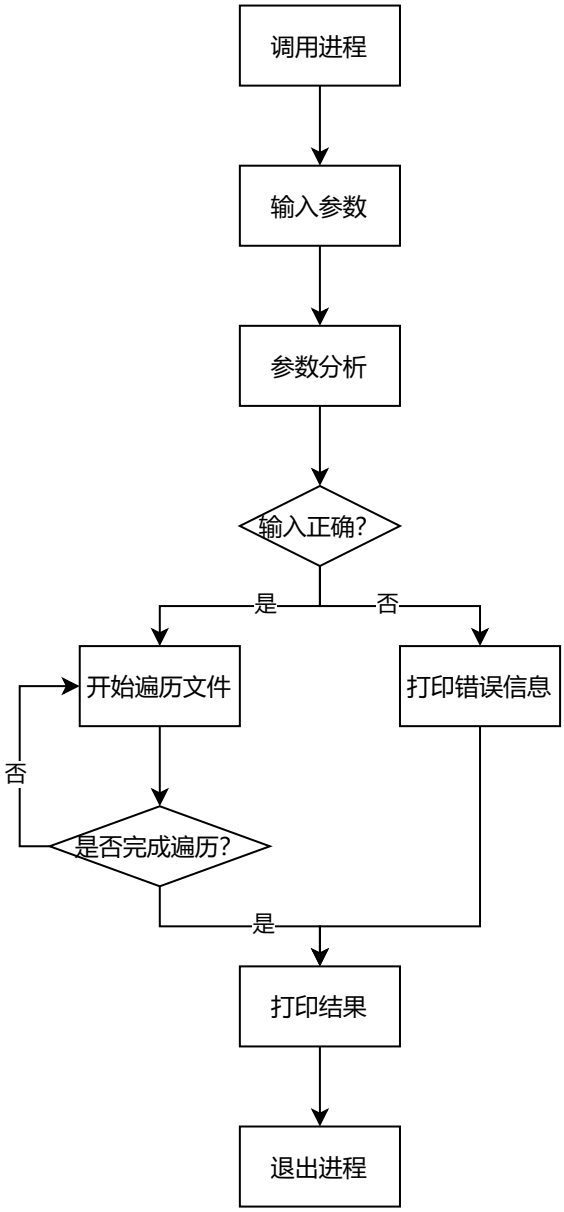
readdir获取一个目录项

- 返回值指针指向的dirent结构体（返回NULL表示已经读到目录尾）
- dirent结构体：记录i节点号和文件名(d_ino和d_name成员)

访问结束：用closedir关闭不再使用的目录句柄。

五、实验步骤

总体流程概述



5.1 参数分析

由于本次作业中，要求不使用函数库中的getopt，自己分割处理命令行的选项，因此我决定参考[GNU libc的getopt函数源码](#)，其中getopt的原型为：

```
#include <unistd.h>
extern char *optarg;  //选项的参数指针
extern int optind,   //下一次调用getopt的时，从optind存储的位置处重新开始检查选项。
extern int opterr,  //当opterr=0时，getopt不向stderr输出错误信息。
extern int optopt;  //当命令行选项字符不包括在optstring中或者选项缺少必要的参数时，该选项存储在optopt中，getopt返回'？'
int getopt(int argc, char * const argv[], const char *optstring);
```

调用一次，返回一个选项。 在命令行选项参数再也检查不到optstring中包含的选项时，返回 - 1，同时optind储存第一个不包含选项的命令行参数。

字符串optstring可以下列元素，

1. 单个字符，表示选项
2. 单个字符后接一个冒号：表示该选项后必须跟一个参数。参数紧跟在选项后或者以空格隔开。该参数的指针赋给optarg。
3. 单个字符后跟两个冒号，表示该选项后必须跟一个参数。参数必须紧跟在选项后不能以空格隔开。该参数的指针赋给optarg。（这个特性是GNU的扩张）。

getopt处理以'-'开头的命令行参数，如optstring="ab:c::d::",命令行为getopt.exe -a -b host -ckeke -d haha
在这个命令行参数中，-a和-h就是选项元素，去掉'-'，a,b,c就是选项。host是b的参数，keke是c的参数。但haha并不是d的参数，因为它们中间有空格隔开。

根据以上原型，自行设计my_getopt函数：

对外数据接口与函数原型：

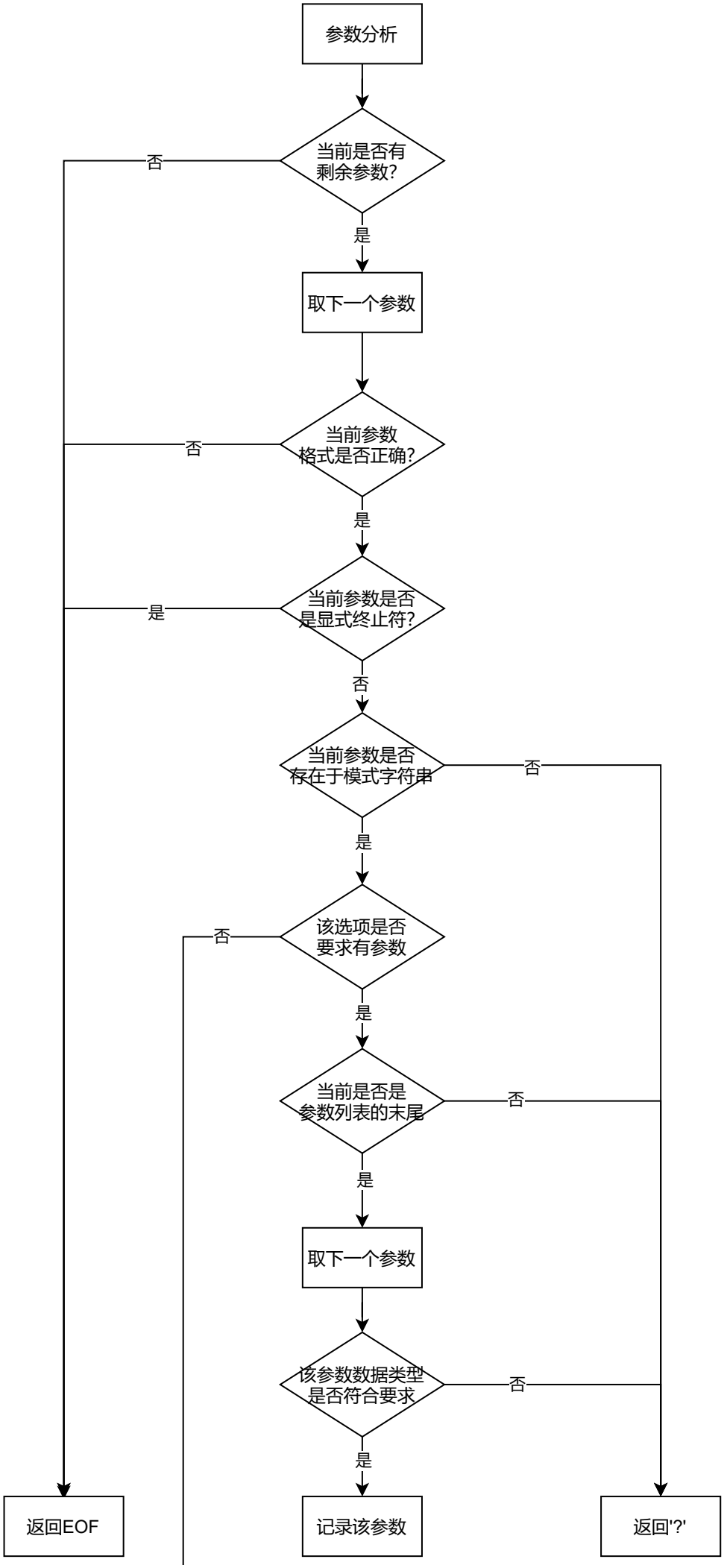
```
extern int optarg; //选项的参数（没有则为-1）
extern int optind; //当前处理的是第几个选项，初始为1（list命令后的第一个参数）
extern int opterr;  //选项错误码，无错误则为0


void get_errmsg(char *msg, char* const argv[]);                    //根据错误码，返回错误信息字符串
int my_getopt(int argc, char* const argv[], const char* optstring); //传入参数个数、参数字符串数组、识别模式，模拟实现getopt的功能
```

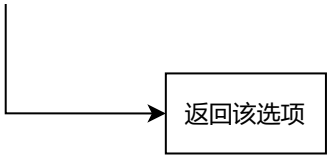
定义错误码：

```
#define INVALID_OPTION 1    //无效选项；错误示例：list -z -> List: invalid option '-z'
#define REQUIRED_OPTION 2   //选项丢失；错误示例：list - -> List: please input option after '-'
#define REQUIRED_ARG 3      //参数缺失；错误示例：list -l -> List: please input argument for '-l'
#define WRONG_TYPE_ARG 4    //参数类型错误；错误示例：list -l a -> List: please input correct argument for '-l'
```

my_getopt处理过程

流程图


参数分析
当前是否有剩余参数?
否
是
取下一个参数
当前参数格式是否正确?
否
是
当前参数是否是显式终止符?
是
否
当前参数是否存在于模式字符串
否
是
该选项是否要求有参数
否
是
当前是否是参数列表的末尾
否
是
取下一个参数
该参数数据类型是否符合要求
否
是
返回EOF
记录该参数
返回'?'



1. 初始化

```
int optchar = EOF;  //选项字符，正确解析时为参数，结束分析或错误时为-1或'?'
optarg = -1;    //初始情况，默认选项无参数
```

2. 结束条件判断

```
if (optind >= argc) {   //当前处理的选项下标已经超过选项总数，说明处理完毕
    return optchar;
}
```

3. 错误的选项格式与显式终止符判断，能够预先判断的错误有`选项前无'-'字符`、`选项丢失(选项只输了一个单独的'-'字符)`；如果识别到`'--'`，说明需要显式地终止命令选项分析。以上几种情况需要直接返回

```
//选项前无'-'
if (argv[optind][0] != '-') {
    return optchar;
}

//选项丢失（选项只输了一个单独的'-'字符）
if (strcmp(argv[optind], "-") == 0) {
    opterr = REQUIRED_OPTION;
    return optchar;
}

//显式地终止命令选项分析
if (strcmp(argv[optind], "--") == 0) {
    optind++;
    return optchar;
}
```

4. 开始分析选项与参数。若选项包含两个以上字符，由于本次编程作业不考虑多选项挤在一个命令行参数内的情况，故不合法。

```
//选项包含两个以上字符，由于本次编程作业不考虑多选项挤在一个命令行参数内的情况，故不合法
if (argv[optind][2] != '\0') {
    opterr = INVALID_OPTION;
    return optchar = '?';
}
```

5. 在模式字符串中搜索当前选项的字符
   - 如果搜索到了该选项，首先判断该位置后有没有`':'`
     - 若有，则需要取下一个参数(`optind++`)，获得该选项的参数；如果此时已经没有下一个参数（`++optind < argc`），说明最后一个需要选项的参数没有输入参数
     - 若没有，则直接判断结束
   - 如果没有在给定选项列表中找到该选项，说明该选项不合法

```
char* optfind = strchr(optstring, optchar);
if (optfind) {
    if (optfind[1] == ':') { //当前搜索得到的模式含':'
         //当前选项需要的参数在下一个argv里,optind+1
        if (++optind < argc) { //如果还有下一个argv
            if ((optarg = trans_int(argv[optind])) == -1) {
                opterr = WRONG_TYPE_ARG;
                return optchar = '?';
            }
        }
        else { //如果已经读完全部argv,说明最后一个需要选项的参数没有输入参数
            opterr = REQUIRED_ARG;
            return optchar = '?';
        }
    }
}
else {  //如果没有在给定选项列表中找到该选项，说明该选项不合法
    opterr = INVALID_OPTION;
    return optchar = '?';
}
```

6. 参数类型转换

   在本次实验中，带有参数的选项只有`-l,-h,-m`三个，且他们需要的参数均为整数，因此引入`trans_int`，将参数字符串转换为预期的数据类型(在本次编程作业中只有整数类型的参数)

```
//将参数字符串转换为预期的数据类型(在本次编程作业中只有整数类型的参数)
int trans_int(char* const arg) {
    for (char* c = arg; *c != '\0'; c++) {
        if (*c < '0' || *c > '9')
            return -1;
    }
    return atoi(arg);
}
```

主程序调用my_getopt过程

定义选项序号

```
#define OPT_A 0
#define OPT_R 1
#define OPT_L 2
#define OPT_H 3
#define OPT_M 4
#define NO_PATH 5   //特殊标记没有输入路径时的情况
```

定义选项状态结构体列表

```
struct OPTION
{
    int flag,    //是否有该选项
        val;    //该选项的参数值
}optlist[OPT_MAXSIZE];
```

调用getopt的过程：每次根据返回的分析结果对相应的选项状态结构体列表赋值。若返回结果为'-1'或 '?'则结束分析

```
//处理输入的选项
while ((c = my_getopt(argc, argv, "ral:h:m:")) != -1) {
    switch (c)
    {
    case 'a':
        optlist[OPT_A].flag = 1;
        break;
    case 'r':
        optlist[OPT_R].flag = 1;
        break;
    case 'l':
        optlist[OPT_L].flag = 1;
```

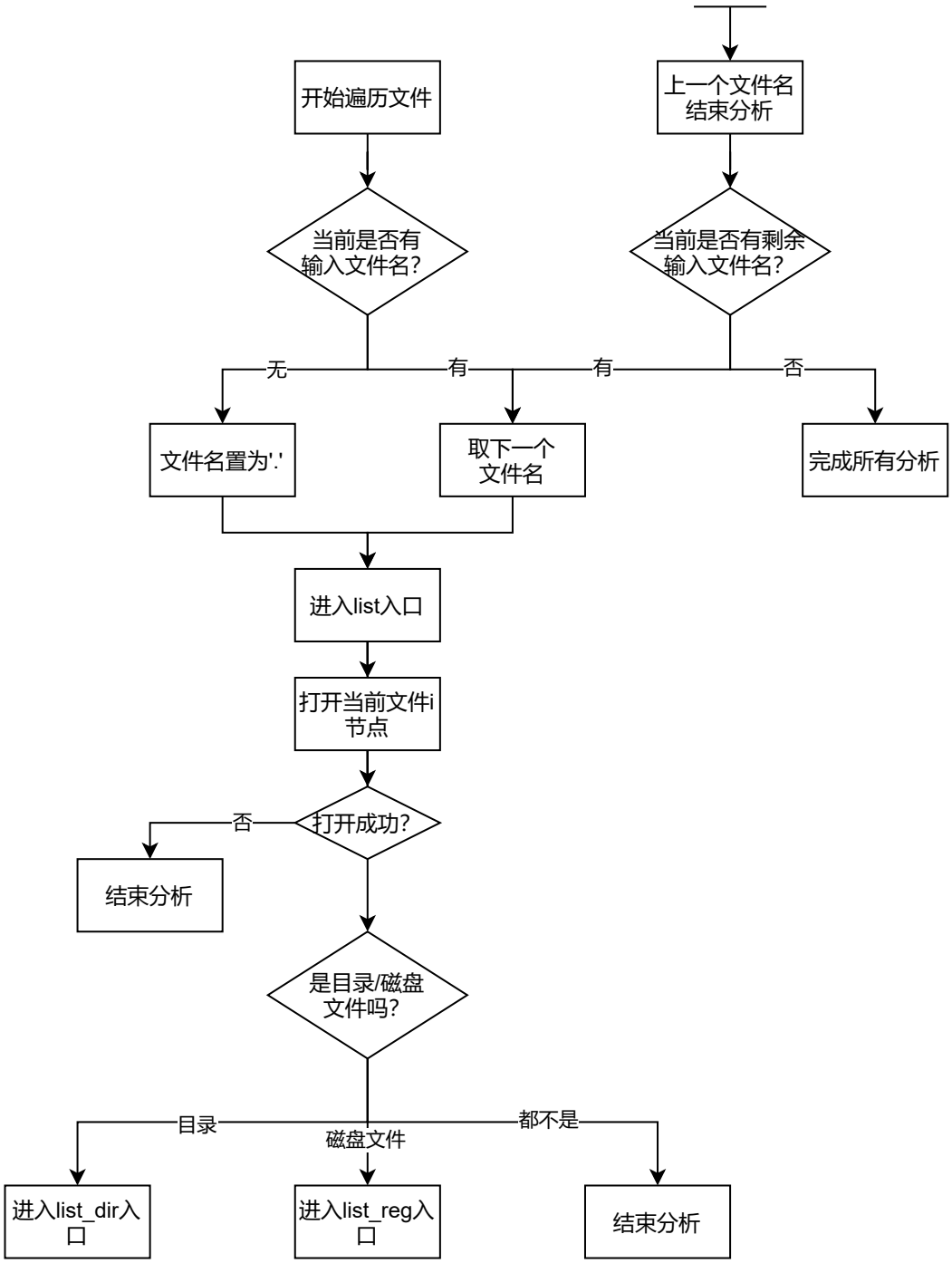
```
        optlist[OPT_L].val = optarg;
        break;
    case 'h':
        optlist[OPT_H].flag = 1;
        optlist[OPT_H].val = optarg;
        break;
    case 'm':
        optlist[OPT_M].flag = 1;
        optlist[OPT_M].val = optarg;
        break;
    default:
        break;
    }
    if (c == '?') break;
}
```

判断解析过程是否出错，如果出错，打印错误信息与帮助信息

```
//选项处理过程中出现错误
if (opterr) {
    char* errmsg = (char*)malloc(ERRMSG_MAXSIZE);
    memset(errmsg, 0, sizeof(errmsg));
    get_errmsg(errmsg, argv);
    printf("List: %s\n", errmsg);
    free(errmsg);
    printf("%s", HELP_MSG);
    return 0;
}
```

5.2 遍历文件

流程图



分析文件名

考虑两种情况：

1. list后没有输入文件名，即分析完参数后optind已经到达结尾，则默认输出当前目录所有文件，传入的文件名为`'.'`，同时标记NO_PATH，表示对`'.'`这个目录打印时不要打印出`./`
2. list后有输入文件名，即分析完参数后optind尚未到达结尾，则依次分析这些文件名

```
if (optind >= argc) {
    optlist[NO_PATH].flag = 1;
    list("", ".");  //默认输出当前目录所有文件
}
//选项后有文件或目录名，遍历之
else
{
    for (int i = optind; i < argc; ++i) {
        list("", argv[i]);
    }
}
```

list主入口

list入口的函数原型如下：

```
void list(const char* prefix, const char* fname)
```

输入路径前缀（递归输出时累积传递）与文件名，该函数判断文件是否为磁盘文件或目录文件，分别调用 `list_reg` 与 `list_dir`

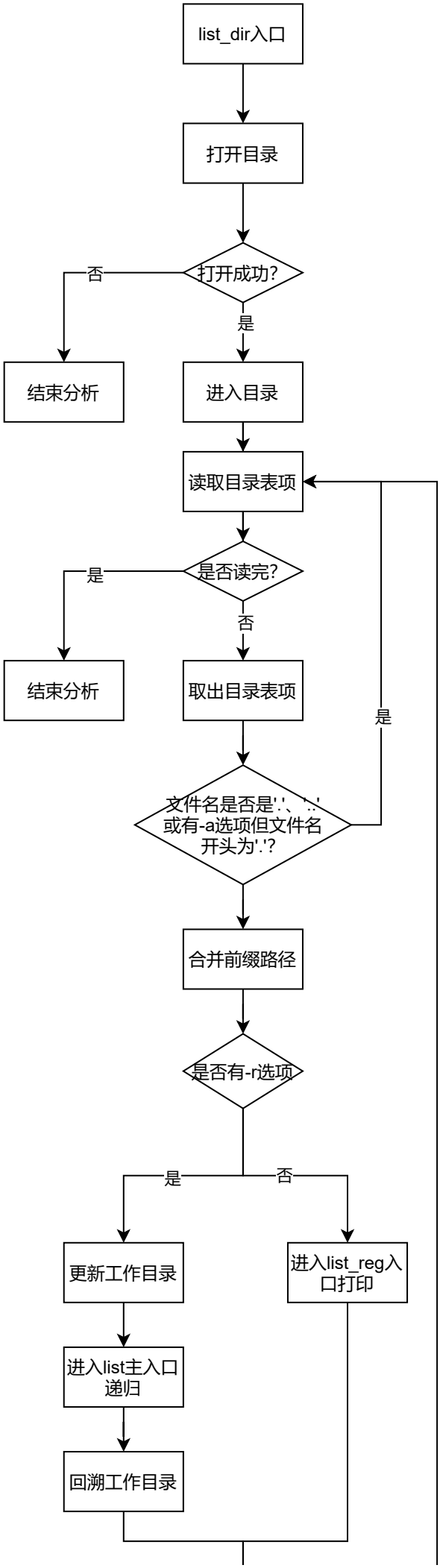
在实现list的函数的过程中，首先调用`stat()`函数获取路径名path对应的i节点中的属性，获取到stat结构体后，通过对 `S_ISDIR` 与 `S_ISREG`函数传入st.st_mode ，可以判断该节点的文件类型是否为磁盘文件或目录文件，而后转到对于入口进一步分析。

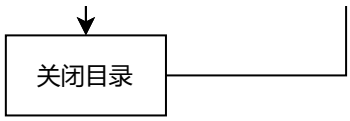
```
//list操作的主函数，输入文件或目录名，输出根据选项处理后文件列表
void list(const char* prefix, const char* fname) {
    //获取路径名path对应的i节点中的属性
    struct stat st;
    if (stat(fname, &st)) {
        printf("List: Can't access \"%s\": %s\n", fname,
strerror(errno));
        return;
    }

    //判断是否为目录
    if (S_ISDIR(st.st_mode)) {
        list_dir(prefix, fname);
    }

    //普通磁盘文件
    else if (S_ISREG(st.st_mode)) {
        list_reg(prefix, fname);
    }
```

```
    //非目录、也非磁盘文件
    else {
        printf("%s 是一个非目录、也非磁盘文件\n", fname);
    }
    return;
}
```

5.3 list_dir目录分析入口

流程图


list_dir入口
打开目录
打开成功?
否
是
结束分析
进入目录
读取目录表项
是否读完?
是
否
结束分析
取出目录表项
是
文件名是否是'.'、'..'
或有-a选项但文件名
开头为'.'?
合并前缀路径
是否有-r选项
是
否
更新工作目录
进入list_reg入
口打印
进入list主入口
递归
回溯工作目录



list_dir目录分析入口的函数原型如下：

```
void list(const char* prefix, const char* fname)
```

在分析目录文件时，首先要通过`opendir()`函数打开目录，通过`readdir()`遍历目录表，获取一个目录项，返回值指针指向的dirent结构体（返回NULL表示已经读到目录尾）。

考虑当前输入是否有`-r`选项：

- 若有`-r`选项，则需要通过`chdir()`函数进入子目录，并通过`rewinddir()`函数更新目录。于此同时，需要将前缀与当前文件名用'/'连接起来。更新完新前缀与新文件名后再次进入list进行递归，最后通过`chdir('..')`返回上一级目录，完成回溯。
- 若没有`-r`选项，则直接将前缀与当前文件名用'/'连接起来，进入list_reg进行普通磁盘文件打印即可。

需要注意的是，在遍历目录表的过程中，如果得到文件名为`.`或`..`的目录项，应当跳过，避免产生死循环。除此之外，在读取到文件名的第一个字符entry_name[0]为`.`时，要判断当前是否有`-a`选项，从而判断是否进入递归或打印。

最后要通过`closedir()`关闭目录。

```
//列出目录文件
void list_dir(const char* prefix, const char* fname) {
    //打开目录
    DIR* dir;
    if ((dir = opendir(fname)) == NULL) {
        printf( "List: Can't access dir \"%s%s\": %s\n", prefix,fname,
strerror(errno));
        return;
    }

    //进入目录
    chdir(fname);
    struct dirent* entry;   //读取目录表项
    char* new_prefix;
    //未输入地址时的特殊处理
    if (optlist[NO_PATH].flag && strcmp(fname, ".") == 0) {
        new_prefix = "";
    }
    else {
```

```
        new_prefix = (char*)malloc(strlen(prefix) + strlen(fname) +
3);   //生成新地址
        pathadd(prefix, fname, new_prefix);
    }

    //有递归选项
    if (optlist[OPT_R].flag) {
        rewinddir(dir);      //更新工作目录
        while ((entry = readdir(dir)) != NULL) {
            const char* entry_name = entry->d_name;
            if (!optlist[OPT_A].flag && entry_name[0] == '.')
                continue;
            if (strcmp(entry_name, ".") == 0 || strcmp(entry_name,
"..") == 0)
                continue;
            list(new_prefix, entry_name);
        }
    }
    //无递归选项
    else {
        while ((entry = readdir(dir)) != NULL) {
            const char* entry_name = entry->d_name;
            if (!optlist[OPT_A].flag && entry_name[0] == '.')
                continue;
            list_reg(new_prefix, entry_name);
        }
    }
    if (closedir(dir)) {
        printf("List: Can't close dir \"%s%s\": %s\n", prefix, fname,
strerror(errno));
        return;
    }
    chdir("..");    //回溯
    return;
}
```

5.4 list_reg磁盘文件分析入口

list_reg磁盘文件分析入口的函数原型如下：

```
void list_reg(const char* prefix, const char* fname)
```

打印磁盘文件的流程相对简单，先通过`stat()`获取i节点属性，然后分别考虑是否有`-l`、是否有`-h`、是否有`-m`，若有则考虑i节点对应属性是否满足要求（有`-m`时用time函数获取系统当前时间），若均满足以上选项，按格式打印即可。

```c
//列出磁盘文件
void list_reg(const char* prefix, const char* fname) {
    //获取路径名path对应的i节点中的属性
    struct stat st;
    time_t nowtime;
    time(&nowtime);
    if (stat(fname, &st)) {
        printf("List: Can't access \"%s\": %s\n", fname,
strerror(errno));
        return;
    }
    off_t fsize = st.st_size;
    time_t fmtime = st.st_mtime;
    if (optlist[OPT_L].flag && fsize < optlist[OPT_L].val ||
        optlist[OPT_H].flag && fsize > optlist[OPT_H].val ||
        optlist[OPT_M].flag && nowtime - fmtime > optlist[OPT_M].val *
24 * 60 * 60)
        return;
    printf("%s\t%10ld\t%s%s\n", (S_ISDIR(st.st_mode) ? "DIR" :
"FILE"), fsize, prefix, fname);
    return;
}
```

5.5 编译

本次实验共有`my_getopt.h`、`my_getopt.c`、`list.c`三个源文件，编写Makefile进行编译：

```makefile
listhty: list.o my_getopt.o
    gcc -Wall list.o my_getopt.o -o listhty

list.o: list.c my_getopt.h
    gcc -c -Wall list.c -o list.o

my_getopt.o: my_getopt.c my_getopt.h
    gcc -c -Wall my_getopt.c -o my_getopt.o

clean:
    rm -rf *.o listhty
```

六、实验结果

1.列出/bin下大小在100~5000之间的文件

```
./listhty -l 100 -h 5000 /bin
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty -l 100 -h 5000 /bin
FILE          1396      /bin/dh_perl_openssl
FILE          1444      /bin/gnome-language-selector
FILE          3662      /bin/jsonpatch
FILE          3592      /bin/run-one-constantly
FILE           397      /bin/jsonschema
FILE          1285      /bin/scsi_start
FILE          1836      /bin/dh_autotools-dev_restoreconfig
FILE          2953      /bin/zipgrep
FILE          1622      /bin/usbip
FILE          1802      /bin/lzless
FILE          2782      /bin/dpkg-distaddfile
FILE          2558      /bin/apport-bug
FILE          2625      /bin/download-mibs
FILE          4566      /bin/debconf-updatepo
FILE          3857      /bin/scsi_satl
FILE          1622      /bin/acpidbg
FILE          4567      /bin/dpkg-parsechangelog
FILE          1455      /bin/ssh-argv0
FILE           126      /bin/gdbtui
FILE          3948      /bin/pod2usage
FILE          3638      /bin/lsb_release
FILE           382      /bin/trial3
FILE          3592      /bin/run-one-until-failure
FILE           946      /bin/which
FILE          4214      /bin/bison.yacc
FILE          4629      /bin/gettext.sh
FILE          2628      /bin/ptardiff
FILE          3592      /bin/run-one-until-success
FILE          1452      /bin/byobu-select-backend
FILE          2346      /bin/gunzip
FILE          4947      /bin/makedumpfile-R.pl
FILE           963      /bin/NF
FILE          2044      /bin/aspell-import
```

2.递归式列出当前目录树下大小超 500B 且 2 天内修改过的文件（包括文件名首字符为圆点的文件）

显然，这两天被修改的文件只有该实验相关的文件

```
./listhty -a -r -l 500 -m 2
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty -a -r -l 500 -m 2
FILE          22120      listhty
FILE           8120      list.o
FILE           4615      my_getopt.c
FILE            851      my_getopt.h
FILE           4392      my_getopt.o
FILE           6944      list.c
```

3.列出文件名为 `-l` 的文件

```
./listhty -- -l
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty -- -l
FILE              0      -l
```

4.list当前目录所有文件

```
./list *
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty *
FILE          30453      beijing
FILE             74      date.awk
FILE            181      final.awk
FILE            866      final.csv
FILE            866      final.csv.backup
FILE             30      get_pmdata.sh
FILE             25      hello.sh
FILE              0      hty/a.txt
DIR            4096      hty/test1
FILE              0      hty/b.txt
FILE              0      -l
FILE          22120      list
FILE           6944      list.c
FILE          22120      listhty
FILE           8120      list.o
FILE            246      makefile
FILE           4615      my_getopt.c
FILE            851      my_getopt.h
FILE           4392      my_getopt.o
FILE            123      pm.awk
FILE             34      test.txt
```

5.列出包含开头为 `.` 的文件

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty -a
FILE          22120      listhty
FILE           8120      list.o
FILE          18144      listhty1
FILE             30      get_pmdata.sh
FILE           4615      my_getopt.c
FILE            866      final.csv
FILE             34      test.txt
FILE              0      -l
FILE             25      hello.sh
FILE            181      final.awk
FILE              0      .a.txt
FILE            123      pm.awk
FILE            866      final.csv.backup
FILE          30453      beijing
FILE            851      my_getopt.h
FILE          22120      list
FILE            246      makefile
FILE           4392      my_getopt.o
FILE             74      date.awk
DIR            4096      hty
FILE           6944      list.c
```

6.递归列出文件

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty -r
FILE          22120      listhty
FILE           8120      list.o
FILE             30      get_pmdata.sh
FILE           4615      my_getopt.c
FILE            866      final.csv
FILE             34      test.txt
FILE              0      -l
FILE             25      hello.sh
FILE            181      final.awk
FILE            123      pm.awk
FILE            866      final.csv.backup
FILE          30453      beijing
FILE            851      my_getopt.h
FILE          22120      list
FILE            246      makefile
FILE           4392      my_getopt.o
FILE             74      date.awk
FILE              0      hty/a.txt
FILE              0      hty/test1/a.txt
FILE              0      hty/test1/b.txt
FILE              0      hty/b.txt
FILE           6944      list.c
```

七、延伸学习

用于处理命令选项的库函数 **getopt_long** 用这个函数重新设计选项处理部分，设计长短格式选项。体会这个库函数功能的设计思想

7.1 原理

getopt函数只能处理短选项，而getopt_long函数两者都可以，可以说getopt_long已经包含了getopt的功能。

getopt族函数一览：

```
#include <unistd.h>
extern char *optarg;
extern int optind, opterr, optopt;
#include <getopt.h>
int getopt(int argc, char * const argv[],const char *optstring);
int getopt_long(int argc, char * const argv[], const char *optstring,
const struct option *longopts, int *longindex);
int getopt_long_only(int argc, char * const argv[], const char
*optstring, const struct option *longopts, int *longindex);
```

参数以及返回值介绍（以上三个函数都适用）：

1、argc和argv和main函数的两个参数一致。

2、optstring: 表示短选项字符串。

```
形式如“a:b::cd:“，分别表示程序支持的命令行短选项有-a、-b、-c、-d，冒号含义如下：
(1)只有一个字符，不带冒号——只表示选项，  如-c
(2)一个字符，后接一个冒号——表示选项后面带一个参数，如-a 100
(3)一个字符，后接两个冒号——表示选项后面带一个可选参数，即参数可有可无，  如果带参数，则选项与参数直接不能有空格
    形式应该如-b200
```

3、longopts：表示长选项结构体。结构如下：

```
struct option
{
     const char *name;
     int         has_arg;
     int        *flag;
     int         val;
};
eg:
```

```
static struct option longOpts[] = {
      { "daemon", no_argument, NULL, 'D' },
      { "dir", required_argument, NULL, 'd' },
      { "out", required_argument, NULL, 'o' },
      { "log", required_argument, NULL, 'l' },
      { "split", required_argument, NULL, 's' },
      { "http-proxy", required_argument, &lopt, 1 },
      { "http-user", required_argument, &lopt, 2 },
      { "http-passwd", required_argument, &lopt, 3 },
      { "http-proxy-user", required_argument, &lopt, 4 },
      { "http-proxy-passwd", required_argument, &lopt, 5 },
      { "http-auth-scheme", required_argument, &lopt, 6 },
      { "version", no_argument, NULL, 'v' },
      { "help", no_argument, NULL, 'h' },
      { 0, 0, 0, 0 }
    };
```

(1)name:表示选项的名称,比如daemon,dir,out等。

(2)has_arg:表示选项后面是否携带参数。该参数有三个不同值，如下：

```
        a:  no_argument(或者是0)时    ——参数后面不跟参数值，eg: --version,--help
        b: required_argument(或者是1)时 ——参数输入格式为：--参数 值 或者 --参数=值。eg:--dir=/home
        c: optional_argument(或者是2)时  ——参数输入格式只能为：--参数=值
```

(3)flag:这个参数有两个意思，空或者非空。

```
        a:如果参数为空NULL，那么当选中某个长选项的时候，getopt_long将返回val值。
                eg，可执行程序 --help，getopt_long的返回值为h.
        b:如果参数不为空，那么当选中某个长选项的时候，getopt_long将返回0，并且将flag指针参数指向val值。

                eg: 可执行程序 --http-proxy=127.0.0.1:80 那么getopt_long返回值为0，并且lopt值为1。
```

(4)val：表示指定函数找到该选项时的返回值，或者当flag非空时指定flag指向的数据的值val。

4、longindex：longindex非空，它指向的变量将记录当前找到参数符合longopts里的第几个元素的描述，即是longopts的下标值。

5、全局变量：

```
（1）optarg：表示当前选项对应的参数值。

（2）optind：表示的是下一个将被处理到的参数在argv中的下标值。

（3）opterr：如果opterr = 0，在getopt、getopt_long、getopt_long_only遇到错误将不会输出错误信息到标准输出流。opterr在非0时，向屏幕输出错误。

（4）optopt：表示没有被未标识的选项。
```

6、返回值：

```
（1）如果短选项找到，那么将返回短选项对应的字符。

（2）如果长选项找到，如果flag为NULL，返回val。如果flag不为空，返回0

（3）如果遇到一个选项没有在短字符、长字符里面。或者在长字符里面存在二义性的，返回“？”

（4）如果解析完所有字符没有找到（一般是输入命令参数格式错误，eg： 连斜杠都没有加的选项），返回“-1”

（5）如果选项需要参数，忘了添加参数。返回值取决于optstring，如果其第一个字符是“：”，则返回“：”，否则返回“？”。
```

注意：

```
（1）longopts的最后一个元素必须是全0填充，否则会报段错误

（2）短选项中每个选项都是唯一的。而长选项如果简写，也需要保持唯一性。
```

7.2 修改代码

```c
//前略
int optIndex = 0;
int lopt;
static struct option longOpts[] = {
  { "all", no_argument, NULL, 'a' },
  { "recursive", no_argument, NULL, 'r' },
  { "low", required_argument, NULL, 'l' },
  { "high", required_argument, NULL, 'h' },
  { "mdays", required_argument, NULL, 'm' },
  { 0, 0, 0, 0 }
};

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

```
    char c = 0;
    int cnt = 0;

    //处理输入的选项
    while ((c = getopt_long(argc, argv, "ral:h:m:", longOpts,
&optIndex)) != -1) {
        switch (c)
        {
        case 'a':
            optlist[OPT_A].flag = 1;
            break;
        case 'r':
            optlist[OPT_R].flag = 1;
            break;
        case 'l':
            optlist[OPT_L].flag = 1;
            optlist[OPT_L].val = atoi(optarg);
            break;
        case 'h':
            optlist[OPT_H].flag = 1;
            optlist[OPT_H].val = atoi(optarg);
            break;
        case 'm':
            optlist[OPT_M].flag = 1;
            optlist[OPT_M].val = atoi(optarg);
            break;
        default:
            break;
        }
    }
//后略
```

7.3 实验效果

1.列出/bin下大小在100~5000之间的文件

```
./listhty1 --low=100 --high=5000 /bin
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty1 --low=100 --high=5000 /bin
FILE          1396      /bin/dh_perl_openssl
FILE          1444      /bin/gnome-language-selector
FILE          3662      /bin/jsonpatch
FILE          3592      /bin/run-one-constantly
FILE           397      /bin/jsonschema
FILE          1285      /bin/scsi_start
FILE          1836      /bin/dh_autotools-dev_restoreconfig
FILE          2953      /bin/zipgrep
FILE          1622      /bin/usbip
FILE          1802      /bin/lzless
FILE          2782      /bin/dpkg-distaddfile
FILE          2558      /bin/apport-bug
FILE          2625      /bin/download-mibs
FILE          4566      /bin/debconf-updatepo
FILE          3857      /bin/scsi_satl
FILE          1622      /bin/acpidbg
FILE          4567      /bin/dpkg-parsechangelog
FILE          1455      /bin/ssh-argv0
FILE           126      /bin/gdbtui
FILE          3948      /bin/pod2usage
FILE          3638      /bin/lsb_release
FILE           382      /bin/trial3
FILE          3592      /bin/run-one-until-failure
FILE           946      /bin/which
FILE          4214      /bin/bison.yacc
FILE          4629      /bin/gettext.sh
FILE          2628      /bin/ptardiff
FILE          3592      /bin/run-one-until-success
FILE          1452      /bin/byobu-select-backend
FILE          2346      /bin/gunzip
FILE          4947      /bin/makedumpfile-R.pl
FILE           963      /bin/NF
```

2.递归式列出当前目录树下大小超 500B 且 2 天内修改过的文件（包括文件名首字符为圆点的文件）

显然，这两天被修改的文件只有该实验相关的文件

```
./listhty1 --all --recursive --low=500 --mdays=2
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty1 --all --recursive --low=500 --mdays=2
FILE         22120      listhty
FILE          8120      list.o
FILE         18144      listhty1
FILE          4615      my_getopt.c
FILE           851      my_getopt.h
FILE          4392      my_getopt.o
FILE          6944      list.c
```

3.递归列出文件

```
./listhty1 --all --recursive
```

```
a0547@Ubuntu-bupt:~/lab2$ ./listhty1 --all --recursive
FILE          22120      listhty
FILE           8120      list.o
FILE          18144      listhty1
FILE             30      get_pmdata.sh
FILE           4615      my_getopt.c
FILE            866      final.csv
FILE             34      test.txt
FILE              0      -l
FILE             25      hello.sh
FILE            181      final.awk
FILE              0      .a.txt
FILE            123      pm.awk
FILE            866      final.csv.backup
FILE          30453      beijing
FILE            851      my_getopt.h
FILE          22120      list
FILE            246      makefile
FILE           4392      my_getopt.o
FILE             74      date.awk
FILE              0      hty/a.txt
FILE              0      hty/test1/a.txt
FILE              0      hty/test1/b.txt
FILE              0      hty/b.txt
FILE           6944      list.c
```

附录 实验代码

my_getopt.h

```
#ifndef MY_GETOPT_H
#define MY_GETOPT_H

#define INVALID_OPTION 1    //无效选项；错误示例：list -z -> List:
invalid option '-z'
#define REQUIRED_OPTION 2   //选项丢失；错误示例：list - -> List: please
input option after '-'
#define REQUIRED_ARG 3      //参数缺失；错误示例：list -l -> List:
please input argument for '-l'
#define WRONG_TYPE_ARG 4    //参数类型错误；错误示例：list -l a -> List:
please input correct argument for '-l'

#define ERRMSG_MAXSIZE 100  //错误信息最大长度

//#define DEBUG

#if defined(__cplusplus)
extern "C" {
#endif

extern int optarg; //选项参数
extern int optind; //当前处理第几个选项
extern int opterr;  //选项错误码，无错误则为0


void get_errmsg(char *msg, char* const argv[]);                    //
根据错误码，返回错误信息字符串
int my_getopt(int argc, char* const argv[], const char* optstring); //
传入参数个数、参数字符串数组、识别模式，模拟实现getopt的功能
#if defined(__cplusplus)
}
#endif

#endif // MY_GETOPT_H
```

my_getopt.c

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

#include "my_getopt.h"


#ifdef DEBUG
void debug(char* const msg) {
    if ((strcmp(msg, "OPTIND") == 0))
        printf("DEBUG: OPTIND=%d\n", optind);
    else
        printf("DEBUG: %s\n", msg);
}
#endif // DEBUG

int optarg;
int optind = 1;
int opterr = 0;



//将参数字符串转换为预期的数据类型(在本次编程作业中只有整数类型的参数)
int trans_int(char* const arg) {
    for (char* c = arg; *c != '\0'; c++) {
#ifdef DEBUG
        printf("DEBUG: 当前待识别是否为整数的参数字符为:%c\n", *c);
#endif // DEBUG

        if (*c < '0' || *c > '9')
            return -1;
    }
    return atoi(arg);
}


int my_getopt(int argc, char* const argv[], const char* optstring) {
    int optchar = EOF;  //选项字符，正确解析时为参数，错误时为'?'

        //判断当前是否处理完所有选项
    if (optind >= argc) {
#ifdef DEBUG
        debug("处理完所有选项,选项后无文件名或目录名");
#endif // DEBUG
        return optchar;
```

```
    }

#ifdef DEBUG
    debug("OPTIND");
#endif // DEBUG

    //初始情况，默认选项无参数
    optarg = -1;

    //选项前无'-'
    if (argv[optind][0] != '-') {
#ifdef DEBUG
        debug("选项前无-,这是一个文件名或目录名,参数处理结束");
#endif // DEBUG

        return optchar;
    }

    //选项丢失（选项只输了一个）
    if (strcmp(argv[optind], "-") == 0) {
#ifdef DEBUG
        debug("选项丢失（选项只输了一个）");
#endif // DEBUG

        opterr = REQUIRED_OPTION;
        return optchar;
    }

    //显式地终止命令选项分析
    if (strcmp(argv[optind], "--") == 0) {
#ifdef DEBUG
        debug("读取到--,显式地终止命令选项分析");
#endif // DEBUG

        optind++;
        return optchar;
    }

    //开始分析选项

    //选项包含两个以上字符，由于本次编程作业不考虑多选项挤在一个命令行参数内的情况，故不合法
    if (argv[optind][2] != '\0') {
#ifdef DEBUG
        debug("选项包含两个以上字符");
#endif // DEBUG
        opterr = INVALID_OPTION;
        return optchar = '?';
    }
```

```
    optchar = argv[optind][1];
#ifdef DEBUG
    printf("DEBUG: 当前识别的选项为 %c\n", optchar);
#endif // DEBUG

    char* optfind = strchr(optstring, optchar);
    if (optfind) {
        if (optfind[1] == ':') { //当前搜索得到的模式含':'
             //当前选项需要的参数在下一个argv里,optind+1
            if (++optind < argc) { //如果还有下一个argv
#ifdef DEBUG
                printf("DEBUG: 当前识别参数为 %s\n", argv[optind]);
#endif // DEBUG
                if ((optarg = trans_int(argv[optind])) == -1) {
#ifdef DEBUG
                    printf("DEBUG: 当前参数'%s'不是一个整数\n",
argv[optind]);
#endif // DEBUG
                    opterr = WRONG_TYPE_ARG;
                    return optchar = '?';
                }
#ifdef DEBUG
                printf("DEBUG: 当前参数'%s'是一个整数\n", argv[optind]);
#endif // DEBUG
            }
            else { //如果已经读完全部argv,说明最后一个需要选项的参数没有输入
参数
#ifdef DEBUG
                debug("最后一个需要选项的参数没有输入参数");
#endif // DEBUG
                opterr = REQUIRED_ARG;
                return optchar = '?';
            }
        }
    }
    else {  //如果没有在给定选项列表中找到该选项，说明该选项不合法
#ifdef DEBUG
        debug("当前选项不合法");
#endif // DEBUG
        opterr = INVALID_OPTION;
        return optchar = '?';
    }

    ++optind;
    return optchar;
}
```

```c
void get_errmsg(char *msg, char* const argv[])
{
    switch (opterr)
    {
    case INVALID_OPTION:    //无效选项
        strcat(msg, "invalid option '");
        strcat(msg, argv[optind]);
        strcat(msg, "'");
        break;
    case REQUIRED_OPTION:   //选项丢失
        strcat(msg, "please input option after '-'");
        break;
    case REQUIRED_ARG:      //参数缺失
        strcat(msg, "please input argument for '");
        strcat(msg, argv[optind - 1]);
        strcat(msg, "'");
        break;
    case WRONG_TYPE_ARG:    //参数类型错误
        strcat(msg, "please input correct argument for '");
        strcat(msg, argv[optind - 1]);
        strcat(msg, "'");
        break;
    default:
        strcat(msg, "unkhnown error");
    }
}
```

list.c

```c
#include <unistd.h>

#include <time.h>
#include <errno.h>
#include <stdio.h>

#include <string.h>
#include <stdlib.h>
#include <dirent.h>

#include <sys/stat.h>
#include <sys/types.h>
#include "my_getopt.h"


//#define DEBUG

//定义选项序号
#define OPT_A 0
#define OPT_R 1
#define OPT_L 2
#define OPT_H 3
#define OPT_M 4
#define NO_PATH 5   //特殊标记没有输入路径时的情况

#define OPT_MAXSIZE 6
#define FNAME_MAXLEN 600

//定义选项状态结构体列表
struct OPTION
{
    int flag,    //是否有该选项
        val;    //该选项的参数值
}optlist[OPT_MAXSIZE];

//定义给用户的帮助信息，提示此命令的用法
const char* HELP_MSG =
"Usage: list1 [OPTION]... [FILE]...\n\
List information about the FILEs (the current directory by default)\n\
\n\
- a           Do not hide entries starting with .\n\
- r           List subdirectories recursively\n\
- l <bytes>   Minimum of file size\n\
- h <bytes>   Maximum of file size\n\
- m <days>    Limit file last modified time\n";
```

```
//组合前缀路径与当前目录名
void pathadd(const char* a, const char* b, char* c) {
    memset(c, 0, sizeof(c));
    strcat(c, a);
    strcat(c, b);
    strcat(c, "/");
    return;
}

//函数原型
void list(const char* prefix, const char* fname);       //list处理主函
数
void list_dir(const char* prefix, const char* fname);   //list处理目录
void list_reg(const char* prefix, const char* fname);   //list处理磁盘
文件

//列出目录文件
void list_dir(const char* prefix, const char* fname) {
#ifdef DEBUG
    printf("DEBUG: list_dir: prefix=%s, fname=%s\n", prefix, fname);
#endif // DEBUG

    //打开目录
    DIR* dir;
    if ((dir = opendir(fname)) == NULL) {
        printf( "List: Can't access dir \"%s%s\": %s\n", prefix,fname,
strerror(errno));
        return;
    }

    //进入目录
    chdir(fname);
    struct dirent* entry;   //读取目录表项
    char* new_prefix;
    //未输入地址时的特殊处理
    if (optlist[NO_PATH].flag && strcmp(fname, ".") == 0) {
        new_prefix = "";
    }
    else {
        new_prefix = (char*)malloc(strlen(prefix) + strlen(fname) +
3);   //生成新地址
        pathadd(prefix, fname, new_prefix);
    }

    //有递归选项
    if (optlist[OPT_R].flag) {
        rewinddir(dir);      //更新工作目录
        while ((entry = readdir(dir)) != NULL) {
            const char* entry_name = entry->d_name;
```

```
            if (!optlist[OPT_A].flag && entry_name[0] == '.')
                continue;
            if (strcmp(entry_name, ".") == 0 || strcmp(entry_name,
"..") == 0)
                continue;
            list(new_prefix, entry_name);
        }
    }
    //无递归选项
    else {
        while ((entry = readdir(dir)) != NULL) {
            const char* entry_name = entry->d_name;
            if (!optlist[OPT_A].flag && entry_name[0] == '.')
                continue;
            list_reg(new_prefix, entry_name);
        }
    }
    if (closedir(dir)) {
        printf("List: Can't close dir \"%s%s\": %s\n", prefix, fname,
strerror(errno));
        return;
    }
    chdir("..");    //回溯
//   if (strcmp(new_prefix, ""))
//       free(new_prefix);
    return;
}

//列出磁盘文件
void list_reg(const char* prefix, const char* fname) {
#ifdef DEBUG
    printf("DEBUG: list_reg: prefix=%s, fname=%s\n", prefix, fname);
#endif // DEBUG

    //获取路径名path对应的i节点中的属性
    struct stat st;
    time_t nowtime;
    time(&nowtime);
    if (stat(fname, &st)) {
        printf("List: Can't access \"%s\": %s\n", fname,
strerror(errno));
        return;
    }
    off_t fsize = st.st_size;
    time_t fmtime = st.st_mtime;
    if (optlist[OPT_L].flag && fsize < optlist[OPT_L].val ||
        optlist[OPT_H].flag && fsize > optlist[OPT_H].val ||
        optlist[OPT_M].flag && nowtime - fmtime > optlist[OPT_M].val *
24 * 60 * 60)
```

```
        return;
    printf("%s\t%10ld\t%s%s\n", (S_ISDIR(st.st_mode) ? "DIR" :
"FILE"), fsize, prefix, fname);
    return;
}

//list操作的主函数，输入文件或目录名，输出根据选项处理后文件列表
void list(const char* prefix, const char* fname) {
#ifdef DEBUG
    printf("DEBUG: list: prefix=%s, fname=%s\n", prefix, fname);
#endif // DEBUG


    //获取路径名path对应的i节点中的属性
    struct stat st;
    if (stat(fname, &st)) {
        printf("List: Can't access \"%s\": %s\n", fname,
strerror(errno));
        return;
    }

    //判断是否为目录
    if (S_ISDIR(st.st_mode)) {
        list_dir(prefix, fname);
    }
    //普通磁盘文件
    else if (S_ISREG(st.st_mode)) {
        list_reg(prefix, fname);
    }
    //非目录、也非磁盘文件
    else {
#ifdef DEBUG
        printf("DEBUG: %s 是一个非目录、也非磁盘文件\n", fname);
#endif // DEBUG
    }
    return;
}

int main(int argc, char** argv)
{

#ifdef DEBUG

    printf("argc = %d, &argc=%lld\n", argc, &argc);
    for (int i = 0; i < argc; ++i)
    {
        printf("argv[%d]=%s\n", i, argv[i]);
    }
#endif // DEBUG
```

```c
    char c = 0;
    int cnt = 0;

    //处理输入的选项
    while ((c = my_getopt(argc, argv, "ral:h:m:")) != -1) {
#ifdef DEBUG
        printf("DEBUG: No.%d option is %c\n", ++cnt, c);
#endif // DEBUG
        switch (c)
        {
        case 'a':
            optlist[OPT_A].flag = 1;
            break;
        case 'r':
            optlist[OPT_R].flag = 1;
            break;
        case 'l':
            optlist[OPT_L].flag = 1;
            optlist[OPT_L].val = optarg;
            break;
        case 'h':
            optlist[OPT_H].flag = 1;
            optlist[OPT_H].val = optarg;
            break;
        case 'm':
            optlist[OPT_M].flag = 1;
            optlist[OPT_M].val = optarg;
            break;
        default:
            break;
        }
    }

    //选项处理过程中出现错误
    if (opterr) {
        char* errmsg = (char*)malloc(ERRMSG_MAXSIZE);
        memset(errmsg, 0, sizeof(errmsg));
        get_errmsg(errmsg, argv);
        printf("List: %s\n", errmsg);
        free(errmsg);
        printf("%s", HELP_MSG);
        return 0;
    }

    //完成选项的解析，准备处理
#ifdef DEBUG
    printf("DEBUG: 选项解析完毕，准备处理\n");
#endif // DEBUG
```

```
    //选项后无文件或目录名，则输出当前目录所有文件
    if (optind >= argc) {
        optlist[NO_PATH].flag = 1;
        list("", ".");  //默认输出当前目录所有文件
    }
    //选项后有文件或目录名，遍历之
    else
    {
        for (int i = optind; i < argc; ++i) {
            list("", argv[i]);
#ifdef DEBUG
            argc = argc_const;
#endif // DEBUG

        }

    }

    return 0;
}
```

makefile

```makefile
listhty: list.o my_getopt.o
    gcc -Wall list.o my_getopt.o -o listhty

list.o: list.c my_getopt.h
    gcc -c -Wall list.c -o list.o

my_getopt.o: my_getopt.c my_getopt.h
    gcc -c -Wall my_getopt.c -o my_getopt.o

clean:
    rm -rf *.o listhty
```

list.c (with getopt_long)

```
#include <unistd.h>

#include <time.h>
#include <errno.h>
#include <stdio.h>

#include <string.h>
#include <stdlib.h>
#include <dirent.h>

#include <sys/stat.h>
#include <sys/types.h>
#include <getopt.h>


//#define DEBUG

//定义选项序号
#define OPT_A 0
#define OPT_R 1
#define OPT_L 2
#define OPT_H 3
#define OPT_M 4
#define NO_PATH 5   //特殊标记没有输入路径时的情况

#define OPT_MAXSIZE 6
#define FNAME_MAXLEN 600

//定义选项状态结构体列表
struct OPTION
{
    int flag,    //是否有该选项
        val;    //该选项的参数值
}optlist[OPT_MAXSIZE];

//定义给用户的帮助信息，提示此命令的用法
const char* HELP_MSG =
"Usage: list1 [OPTION]... [FILE]...\n\
List information about the FILEs (the current directory by default)\n\
\n\
- a           Do not hide entries starting with .\n\
- r           List subdirectories recursively\n\
- l <bytes>   Minimum of file size\n\
- h <bytes>   Maximum of file size\n\
- m <days>    Limit file last modified time\n";
```

```
//组合前缀路径与当前目录名
void pathadd(const char* a, const char* b, char* c) {
    memset(c, 0, sizeof(c));
    strcat(c, a);
    strcat(c, b);
    strcat(c, "/");
    return;
}

//函数原型
void list(const char* prefix, const char* fname);       //list处理主函
数
void list_dir(const char* prefix, const char* fname);   //list处理目录
void list_reg(const char* prefix, const char* fname);   //list处理磁盘
文件

//列出目录文件
void list_dir(const char* prefix, const char* fname) {
#ifdef DEBUG
    printf("DEBUG: list_dir: prefix=%s, fname=%s\n", prefix, fname);
#endif // DEBUG

    //打开目录
    DIR* dir;
    if ((dir = opendir(fname)) == NULL) {
        printf( "List: Can't access dir \"%s%s\": %s\n", prefix,fname,
strerror(errno));
        return;
    }

    //进入目录
    chdir(fname);
    struct dirent* entry;   //读取目录表项
    char* new_prefix;
    //未输入地址时的特殊处理
    if (optlist[NO_PATH].flag && strcmp(fname, ".") == 0) {
        new_prefix = "";
    }
    else {
        new_prefix = (char*)malloc(strlen(prefix) + strlen(fname) +
3);   //生成新地址
        pathadd(prefix, fname, new_prefix);
    }

    //有递归选项
    if (optlist[OPT_R].flag) {
        rewinddir(dir);      //更新工作目录
        while ((entry = readdir(dir)) != NULL) {
            const char* entry_name = entry->d_name;
```

```
            if (!optlist[OPT_A].flag && entry_name[0] == '.')
                continue;
            if (strcmp(entry_name, ".") == 0 || strcmp(entry_name,
"..") == 0)
                continue;
            list(new_prefix, entry_name);
        }
    }
    //无递归选项
    else {
        while ((entry = readdir(dir)) != NULL) {
            const char* entry_name = entry->d_name;
            if (!optlist[OPT_A].flag && entry_name[0] == '.')
                continue;
            list_reg(new_prefix, entry_name);
        }
    }
    if (closedir(dir)) {
        printf("List: Can't close dir \"%s%s\": %s\n", prefix, fname,
strerror(errno));
        return;
    }
    chdir("..");    //回溯
//   if (strcmp(new_prefix, ""))
//       free(new_prefix);
    return;
}

//列出磁盘文件
void list_reg(const char* prefix, const char* fname) {
#ifdef DEBUG
    printf("DEBUG: list_reg: prefix=%s, fname=%s\n", prefix, fname);
#endif // DEBUG

    //获取路径名path对应的i节点中的属性
    struct stat st;
    time_t nowtime;
    time(&nowtime);
    if (stat(fname, &st)) {
        printf("List: Can't access \"%s\": %s\n", fname,
strerror(errno));
        return;
    }
    off_t fsize = st.st_size;
    time_t fmtime = st.st_mtime;
    if (optlist[OPT_L].flag && fsize < optlist[OPT_L].val ||
        optlist[OPT_H].flag && fsize > optlist[OPT_H].val ||
        optlist[OPT_M].flag && nowtime - fmtime > optlist[OPT_M].val *
24 * 60 * 60)
```

```
        return;
    printf("%s\t%10ld\t%s%s\n", (S_ISDIR(st.st_mode) ? "DIR" :
"FILE"), fsize, prefix, fname);
    return;
}

//list操作的主函数，输入文件或目录名，输出根据选项处理后文件列表
void list(const char* prefix, const char* fname) {
#ifdef DEBUG
    printf("DEBUG: list: prefix=%s, fname=%s\n", prefix, fname);
#endif // DEBUG


    //获取路径名path对应的i节点中的属性
    struct stat st;
    if (stat(fname, &st)) {
        printf("List: Can't access \"%s\": %s\n", fname,
strerror(errno));
        return;
    }

    //判断是否为目录
    if (S_ISDIR(st.st_mode)) {
        list_dir(prefix, fname);
    }
    //普通磁盘文件
    else if (S_ISREG(st.st_mode)) {
        list_reg(prefix, fname);
    }
    //非目录、也非磁盘文件
    else {
#ifdef DEBUG
        printf("DEBUG: %s 是一个非目录、也非磁盘文件\n", fname);
#endif // DEBUG
    }
    return;
}

int optIndex = 0;
int lopt;
static struct option longOpts[] = {
  { "all", no_argument, NULL, 'a' },
  { "recursive", no_argument, NULL, 'r' },
  { "low", required_argument, NULL, 'l' },
  { "high", required_argument, NULL, 'h' },
  { "mdays", required_argument, NULL, 'm' },
  { 0, 0, 0, 0 }
};
```

```c
int main(int argc, char** argv)
{

#ifdef DEBUG

    printf("argc = %d, &argc=%lld\n", argc, &argc);
    for (int i = 0; i < argc; ++i)
    {
        printf("argv[%d]=%s\n", i, argv[i]);
    }
#endif // DEBUG

    char c = 0;
    int cnt = 0;

    //处理输入的选项
    while ((c = getopt_long(argc, argv, "ral:h:m:", longOpts,
&optIndex)) != -1) {
#ifdef DEBUG
        printf("DEBUG: No.%d option is %c\n", ++cnt, c);
#endif // DEBUG
        switch (c)
        {
        case 'a':
            optlist[OPT_A].flag = 1;
            break;
        case 'r':
            optlist[OPT_R].flag = 1;
            break;
        case 'l':
            optlist[OPT_L].flag = 1;
            optlist[OPT_L].val = atoi(optarg);
            break;
        case 'h':
            optlist[OPT_H].flag = 1;
            optlist[OPT_H].val = atoi(optarg);
            break;
        case 'm':
            optlist[OPT_M].flag = 1;
            optlist[OPT_M].val = atoi(optarg);
            break;
        default:
            break;
        }
    }

    //完成选项的解析，准备处理
#ifdef DEBUG
    printf("DEBUG: 选项解析完毕，准备处理\n");
```

```
#endif // DEBUG
    //选项后无文件或目录名，则输出当前目录所有文件
    if (optind >= argc) {
        optlist[NO_PATH].flag = 1;
        list("", ".");  //默认输出当前目录所有文件
    }
    //选项后有文件或目录名，遍历之
    else
    {
        for (int i = optind; i < argc; ++i) {
            list("", argv[i]);
#ifdef DEBUG
            argc = argc_const;
#endif // DEBUG

        }

    }

    return 0;
}
```

参考资料

getopt（From GNU libc源码）

getopt.h

```
/* Declarations for getopt.
   Copyright (C) 1989,90,91,92,93,94,96,97 Free Software Foundation,
Inc.
   NOTE: The canonical source of this file is maintained with the GNU
C Library.
   Bugs can be reported to bug-glibc@gnu.org.
   This program is free software; you can redistribute it and/or
modify it
   under the terms of the GNU General Public License as published by
the
   Free Software Foundation; either version 2, or (at your option) any
   later version.
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-
1307,
   USA.  */

#ifndef _GETOPT_H
#define _GETOPT_H 1

#ifdef  __cplusplus
extern "C" {
#endif

/* For communication from `getopt' to the caller.
   When `getopt' finds an option that takes an argument,
   the argument value is returned here.
   Also, when `ordering' is RETURN_IN_ORDER,
   each non-option ARGV-element is returned here.  */

extern char *optarg;

/* Index in ARGV of the next element to be scanned.
   This is used for communication to and from the caller
   and for communication between successive calls to `getopt'.
```

```
   On entry to `getopt', zero means this is the first call;
initialize.
   When `getopt' returns -1, this is the index of the first of the
   non-option elements that the caller should itself scan.
   Otherwise, `optind' communicates from one call to the next
   how much of ARGV has been scanned so far.  */

extern int optind;

/* Callers store zero here to inhibit the error message `getopt'
prints
   for unrecognized options.  */

extern int opterr;

/* Set to an option character which was unrecognized.  */

extern int optopt;

/* Describe the long-named options requested by the application.
   The LONG_OPTIONS argument to getopt_long or getopt_long_only is a
vector
   of `struct option' terminated by an element containing a name which
is
   zero.
   The field `has_arg' is:
   no_argument      (or 0) if the option does not take an argument,
   required_argument    (or 1) if the option requires an argument,
   optional_argument    (or 2) if the option takes an optional
argument.
   If the field `flag' is not NULL, it points to a variable that is
set
   to the value given in the field `val' when the option is found, but
   left unchanged if the option is not found.
   To have a long-named option do something other than set an `int' to
   a compiled-in constant, such as set a value from `optarg', set the
   option's `flag' field to zero and its `val' field to a nonzero
   value (the equivalent single-letter option character, if there is
   one).  For long options that have a zero `flag' field, `getopt'
   returns the contents of the `val' field.  */

struct option
{
#if defined (__STDC__) && __STDC__
  const char *name;
#else
  char *name;
#endif
  /* has_arg can't be an enum because some compilers complain about
```

```
     type mismatches in all the code that assumes it is an int.  */
  int has_arg;
  int *flag;
  int val;
};

/* Names for the values of the `has_arg' field of `struct option'.  */

#define no_argument     0
#define required_argument   1
#define optional_argument   2

#if defined (__STDC__) && __STDC__
#ifdef __GNU_LIBRARY__
/* Many other libraries have conflicting prototypes for getopt, with
   differences in the consts, in stdlib.h.  To avoid compilation
   errors, only prototype getopt for the GNU C library.  */
extern int getopt (int argc, char *const *argv, const char
*shortopts);
#else /* not __GNU_LIBRARY__ */
extern int getopt ();
#endif /* __GNU_LIBRARY__ */
extern int getopt_long (int argc, char *const *argv, const char
*shortopts,
                const struct option *longopts, int *longind);
extern int getopt_long_only (int argc, char *const *argv,
                 const char *shortopts,
                     const struct option *longopts, int *longind);

/* Internal only.  Users should not call this directly.  */
extern int _getopt_internal (int argc, char *const *argv,
                 const char *shortopts,
                     const struct option *longopts, int *longind,
                 int long_only);
#else /* not __STDC__ */
extern int getopt ();
extern int getopt_long ();
extern int getopt_long_only ();

extern int _getopt_internal ();
#endif /* __STDC__ */

#ifdef  __cplusplus
}
#endif

#endif /* getopt.h */
```

getopt.c

```
/*  $OpenBSD: getopt_long.c,v 1.20 2005/10/25 15:49:37 jmc Exp $    */
/*  $NetBSD: getopt_long.c,v 1.15 2002/01/31 22:43:40 tv Exp $  */

/*
 * Copyright (c) 2002 Todd C. Miller <Todd.Miller@courtesan.com>
 *
 * Permission to use, copy, modify, and distribute this software for
any
 * purpose with or without fee is hereby granted, provided that the
above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE
FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 * Sponsored in part by the Defense Advanced Research Projects
 * Agency (DARPA) and Air Force Research Laboratory, Air Force
 * Materiel Command, USAF, under agreement number F39502-99-1-0512.
 */
/*-
 * Copyright (c) 2000 The NetBSD Foundation, Inc.
 * All rights reserved.
 *
 * This code is derived from software contributed to The NetBSD
Foundation
 * by Dieter Baron and Thomas Klausner.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above
copyright
 *    notice, this list of conditions and the following disclaimer in
the
```

```
 *    documentation and/or other materials provided with the
distribution.
 * 3. All advertising materials mentioning features or use of this
software
 *    must display the following acknowledgement:
 *        This product includes software developed by the NetBSD
 *        Foundation, Inc. and its contributors.
 * 4. Neither the name of The NetBSD Foundation nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC. AND
CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE FOUNDATION OR
CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#define _CRT_SECURE_NO_WARNINGS 1

#include <errno.h>
#include "getopt.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define REPLACE_GETOPT      /* use this getopt as the system getopt(3)
*/

#ifdef REPLACE_GETOPT
int opterr = 1;     /* if error message should be printed */
int optind = 1;     /* index into parent argv vector */
int optopt = '?';       /* character checked for validity */
```

```
char    *optarg;        /* argument associated with option */
#endif
int optreset;       /* reset getopt */

#define PRINT_ERROR ((opterr) && (*options != ':'))

#define FLAG_PERMUTE    0x01    /* permute non-options to the end of
argv */
#define FLAG_ALLARGS    0x02    /* treat non-options as args to option
"-1" */
#define FLAG_LONGONLY   0x04    /* operate as getopt_long_only */

/* return values */
#define BADCH       (int)'?'
#define BADARG      ((*options == ':') ? (int)':' : (int)'?')
#define INORDER     (int)1

#define EMSG        ""

static int getopt_internal(int, char * const *, const char *,
               const struct option *, int *, int);
static int parse_long_options(char * const *, const char *,
                  const struct option *, int *, int);
static int gcd(int, int);
static void permute_args(int, int, int, char * const *);

static char *place = EMSG; /* option letter processing */

/* XXX: set optreset to 1 rather than these two */
static int nonopt_start = -1; /* first non option argument (for
permute) */
static int nonopt_end = -1;   /* first option after non options (for
permute) */

/* Error messages */
static const char recargchar[] = "option requires an argument -- %c";
static const char recargstring[] = "option requires an argument --
%s";
static const char ambig[] = "ambiguous option -- %.*s";
static const char noarg[] = "option doesn't take an argument -- %.*s";
static const char illoptchar[] = "unknown option -- %c";
static const char illoptstring[] = "unknown option -- %s";

/*
 * Compute the greatest common divisor of a and b.
 */
static int
gcd(int a, int b)
{
```

```c
    int c;

    c = a % b;
    while (c != 0) {
        a = b;
        b = c;
        c = a % b;
    }

    return (b);
}

/*
 * Exchange the block from nonopt_start to nonopt_end with the block
 * from nonopt_end to opt_end (keeping the same order of arguments
 * in each block).
 */
static void
permute_args(int panonopt_start, int panonopt_end, int opt_end,
    char * const *nargv)
{
    int cstart, cyclelen, i, j, ncycle, nnonopts, nopts, pos;
    char *swap;

    /*
     * compute lengths of blocks and number and size of cycles
     */
    nnonopts = panonopt_end - panonopt_start;
    nopts = opt_end - panonopt_end;
    ncycle = gcd(nnonopts, nopts);
    cyclelen = (opt_end - panonopt_start) / ncycle;

    for (i = 0; i < ncycle; i++) {
        cstart = panonopt_end+i;
        pos = cstart;
        for (j = 0; j < cyclelen; j++) {
            if (pos >= panonopt_end)
                pos -= nnonopts;
            else
                pos += nopts;
            swap = nargv[pos];
            /* LINTED const cast */
            ((char **) nargv)[pos] = nargv[cstart];
            /* LINTED const cast */
            ((char **)nargv)[cstart] = swap;
        }
    }
}
```

```c
/*
 * parse_long_options --
 *  Parse long options in argc/argv argument vector.
 * Returns -1 if short_too is set and the option does not match
long_options.
 */
static int
parse_long_options(char * const *nargv, const char *options,
    const struct option *long_options, int *idx, int short_too)
{
    char *current_argv, *has_equal;
    size_t current_argv_len;
    int i, match;

    current_argv = place;
    match = -1;

    optind++;

    if ((has_equal = strchr(current_argv, '=')) != NULL) {
        /* argument found (--option=arg) */
        current_argv_len = has_equal - current_argv;
        has_equal++;
    } else
        current_argv_len = strlen(current_argv);

    for (i = 0; long_options[i].name; i++) {
        /* find matching long option */
        if (strncmp(current_argv, long_options[i].name,
            current_argv_len))
            continue;

        if (strlen(long_options[i].name) == current_argv_len) {
            /* exact match */
            match = i;
            break;
        }
        /*
         * If this is a known short option, don't allow
         * a partial match of a single character.
         */
        if (short_too && current_argv_len == 1)
            continue;

        if (match == -1)    /* partial match */
            match = i;
        else {
            /* ambiguous abbreviation */
            if (PRINT_ERROR)
```

```
                fprintf(stderr,
                        ambig, (int)current_argv_len,
                        current_argv);
            optopt = 0;
            return (BADCH);
        }
    }
    if (match != -1) {      /* option found */
        if (long_options[match].has_arg == no_argument
            && has_equal) {
            if (PRINT_ERROR)
                fprintf(stderr,
                        noarg, (int)current_argv_len,
                        current_argv);
            /*
             * XXX: GNU sets optopt to val regardless of flag
             */
            if (long_options[match].flag == NULL)
                optopt = long_options[match].val;
            else
                optopt = 0;
            return (BADARG);
        }
        if (long_options[match].has_arg == required_argument ||
            long_options[match].has_arg == optional_argument) {
            if (has_equal)
                optarg = has_equal;
            else if (long_options[match].has_arg ==
                required_argument) {
                /*
                 * optional argument doesn't use next nargv
                 */
                optarg = nargv[optind++];
            }
        }
        if ((long_options[match].has_arg == required_argument)
            && (optarg == NULL)) {
            /*
             * Missing argument; leading ':' indicates no error
             * should be generated.
             */
            if (PRINT_ERROR)
                fprintf(stderr,
                        recargstring,
                        current_argv);
            /*
             * XXX: GNU sets optopt to val regardless of flag
             */
            if (long_options[match].flag == NULL)
```

```
                optopt = long_options[match].val;
            else
                optopt = 0;
            --optind;
            return (BADARG);
        }
    } else {            /* unknown option */
        if (short_too) {
            --optind;
            return (-1);
        }
        if (PRINT_ERROR)
            fprintf(stderr, illoptstring, current_argv);
        optopt = 0;
        return (BADCH);
    }
    if (idx)
        *idx = match;
    if (long_options[match].flag) {
        *long_options[match].flag = long_options[match].val;
        return (0);
    } else
        return (long_options[match].val);
}

/*
 * getopt_internal --
 *  Parse argc/argv argument vector.  Called by user level routines.
 */
static int
getopt_internal(int nargc, char * const *nargv, const char *options,
    const struct option *long_options, int *idx, int flags)
{
    char *oli;              /* option letter list index */
    int optchar, short_too;
    static int posixly_correct = -1;

    if (options == NULL)
        return (-1);

    /*
     * Disable GNU extensions if POSIXLY_CORRECT is set or options
     * string begins with a '+'.
     */
    if (posixly_correct == -1)
        posixly_correct = (getenv("POSIXLY_CORRECT") != NULL);
    if (posixly_correct || *options == '+')
        flags &= ~FLAG_PERMUTE;
    else if (*options == '-')
```

```
        flags |= FLAG_ALLARGS;
    if (*options == '+' || *options == '-')
        options++;

    /*
     * XXX Some GNU programs (like cvs) set optind to 0 instead of
     * XXX using optreset.  Work around this braindamage.
     */
    if (optind == 0)
        optind = optreset = 1;

    optarg = NULL;
    if (optreset)
        nonopt_start = nonopt_end = -1;
start:
    if (optreset || !*place) {      /* update scanning pointer */
        optreset = 0;
        if (optind >= nargc) {          /* end of argument vector */
            place = EMSG;
            if (nonopt_end != -1) {
                /* do permutation, if we have to */
                permute_args(nonopt_start, nonopt_end,
                    optind, nargv);
                optind -= nonopt_end - nonopt_start;
            }
            else if (nonopt_start != -1) {
                /*
                 * If we skipped non-options, set optind
                 * to the first of them.
                 */
                optind = nonopt_start;
            }
            nonopt_start = nonopt_end = -1;
            return (-1);
        }
        if (*(place = nargv[optind]) != '-' ||
            (place[1] == '\0' && strchr(options, '-') == NULL)) {
            place = EMSG;       /* found non-option */
            if (flags & FLAG_ALLARGS) {
                /*
                 * GNU extension:
                 * return non-option as argument to option 1
                 */
                optarg = nargv[optind++];
                return (INORDER);
            }
            if (!(flags & FLAG_PERMUTE)) {
                /*
                 * If no permutation wanted, stop parsing
```

```
                 * at first non-option.
                 */
                return (-1);
            }
            /* do permutation */
            if (nonopt_start == -1)
                nonopt_start = optind;
            else if (nonopt_end != -1) {
                permute_args(nonopt_start, nonopt_end,
                    optind, nargv);
                nonopt_start = optind -
                    (nonopt_end - nonopt_start);
                nonopt_end = -1;
            }
            optind++;
            /* process next argument */
            goto start;
        }
        if (nonopt_start != -1 && nonopt_end == -1)
            nonopt_end = optind;

        /*
         * If we have "-" do nothing, if "--" we are done.
         */
        if (place[1] != '\0' && *++place == '-' && place[1] == '\0') {
            optind++;
            place = EMSG;
            /*
             * We found an option (--), so if we skipped
             * non-options, we have to permute.
             */
            if (nonopt_end != -1) {
                permute_args(nonopt_start, nonopt_end,
                    optind, nargv);
                optind -= nonopt_end - nonopt_start;
            }
            nonopt_start = nonopt_end = -1;
            return (-1);
        }
    }

    /*
     * Check long options if:
     *  1) we were passed some
     *  2) the arg is not just "-"
     *  3) either the arg starts with -- we are getopt_long_only()
     */
    if (long_options != NULL && place != nargv[optind] &&
        (*place == '-' || (flags & FLAG_LONGONLY))) {
```

```
        short_too = 0;
        if (*place == '-')
            place++;        /* --foo long option */
        else if (*place != ':' && strchr(options, *place) != NULL)
            short_too = 1;      /* could be short option too */

        optchar = parse_long_options(nargv, options, long_options,
            idx, short_too);
        if (optchar != -1) {
            place = EMSG;
            return (optchar);
        }
    }

    if (((optchar = (int)*place++) == (int)':') ||
        (optchar == (int)'-' && *place != '\0') ||
        (oli = strchr(options, optchar)) == NULL) {
        /*
         * If the user specified "-" and  '-' isn't listed in
         * options, return -1 (non-option) as per POSIX.
         * Otherwise, it is an unknown option character (or ':').
         */
        if (optchar == (int)'-' && *place == '\0')
            return (-1);
        if (!*place)
            ++optind;
        if (PRINT_ERROR)
            fprintf(stderr, illoptchar, optchar);
        optopt = optchar;
        return (BADCH);
    }
    if (long_options != NULL && optchar == 'W' && oli[1] == ';') {
        /* -W long-option */
        if (*place)         /* no space */
            /* NOTHING */;
        else if (++optind >= nargc) {   /* no arg */
            place = EMSG;
            if (PRINT_ERROR)
                fprintf(stderr, recargchar, optchar);
            optopt = optchar;
            return (BADARG);
        } else              /* white space */
            place = nargv[optind];
        optchar = parse_long_options(nargv, options, long_options,
            idx, 0);
        place = EMSG;
        return (optchar);
    }
    if (*++oli != ':') {            /* doesn't take argument */
```

```
        if (!*place)
            ++optind;
    } else {                /* takes (optional) argument */
        optarg = NULL;
        if (*place)         /* no white space */
            optarg = place;
        /* XXX: disable test for :: if PC? (GNU doesn't) */
        else if (oli[1] != ':') {   /* arg not optional */
            if (++optind >= nargc) {    /* no arg */
                place = EMSG;
                if (PRINT_ERROR)
                    fprintf(stderr, recargchar, optchar);
                optopt = optchar;
                return (BADARG);
            } else
                optarg = nargv[optind];
        } else if (!(flags & FLAG_PERMUTE)) {
            /*
             * If permutation is disabled, we can accept an
             * optional arg separated by whitespace so long
             * as it does not start with a dash (-).
             */
            if (optind + 1 < nargc && *nargv[optind + 1] != '-')
                optarg = nargv[++optind];
        }
        place = EMSG;
        ++optind;
    }
    /* dump back option letter */
    return (optchar);
}

#ifdef REPLACE_GETOPT
/*
 * getopt --
 *  Parse argc/argv argument vector.
 *
 * [eventually this will replace the BSD getopt]
 */
int
getopt(int nargc, char * const *nargv, const char *options)
{

    /*
     * We don't pass FLAG_PERMUTE to getopt_internal() since
     * the BSD getopt(3) (unlike GNU) has never done this.
     *
     * Furthermore, since many privileged programs call getopt()
     * before dropping privileges it makes sense to keep things
```

```
     * as simple (and bug-free) as possible.
     */
    return (getopt_internal(nargc, nargv, options, NULL, NULL, 0));
}
#endif /* REPLACE_GETOPT */

/*
 * getopt_long --
 *  Parse argc/argv argument vector.
 */
int
getopt_long(int nargc, char * const *nargv, const char *options,
    const struct option *long_options, int *idx)
{

    return (getopt_internal(nargc, nargv, options, long_options, idx,
        FLAG_PERMUTE));
}

/*
 * getopt_long_only --
 *  Parse argc/argv argument vector.
 */
int
getopt_long_only(int nargc, char * const *nargv, const char *options,
    const struct option *long_options, int *idx)
{

    return (getopt_internal(nargc, nargv, options, long_options, idx,
        FLAG_PERMUTE|FLAG_LONGONLY));
}

/*****************************************************************
********
* getopt test program.
* $Header: /cvsroot/freegetopt/freegetopt/test.c,v 1.3 2003/10/26
03:10:20 vindaci Exp $
*
* Copyright (c)2002-2003 Mark K. Kim
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*
```

```
*   * Redistributions in binary form must reproduce the above
copyright
*     notice, this list of conditions and the following disclaimer in
*     the documentation and/or other materials provided with the
*     distribution.
*
*   * Neither the original author of this software nor the names of
its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS
* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF
* THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
* DAMAGE.
*/


#define _CRT_SECURE_NO_WARNINGS 1


#include <stdio.h>
#include <stdlib.h>
#include "getopt.h"


/*********************************************************************
********
* DEFINES
*/

/**
* flags for different command-line options
*
* these options don't do anything - there's just here
* as examples
*/
#define FLAG_INTERACT   0x0001         /* interactive mode */
```

```c
#define FLAG_FORCE      0x0002          /* force mode */
#define FLAG_RECURSIVE  0x0004          /* recursive mode */


/*********************************************************************
********
* GLOBALS
*/

int flags = 0;                          /* store flags here */

int verbose = 5;                        /* verbosity level */
const char* in_fname = NULL;            /* input filename */
const char* out_fname = NULL;           /* output filename */


/*********************************************************************
********
* arg_to_int - Convert argument string to integer.
*
* min - Minimum allowed value, inclusive.
* max - Maximum allowed value, inclusive.
* defalt - The default value, in case of an error.
* opt - Option string of this argument.  (ex., "-h");
*/

int arg_to_int(const char* arg, int min, int max, int defalt, const
char* opt)
{
    int i = defalt;
    int rv;

    /* no argument means we use the default value */
    if(!arg) goto done;

    /* make sure we got an integer argument */
    rv = sscanf(arg, "%d", &i);
    if(rv != 1) {
        fprintf(stderr, "%s: integer argument required.\n", opt);
        i = defalt;
        goto done;
    }

    /* make sure the integer argument is within the desired range */
    if(i < min || max < i) {
        fprintf(stderr, "%s: argument out of integer range.\n", opt);
        i = defalt;
        goto done;
    }
```

```
done:
    return i;
}


/*********************************************************************
********
* help
*/

void help()
{
    printf(
"getopt test program\n"
"Usage: test [OPTION] [INPUT]\n"
"   INPUT           set input filename (doesn't do anything)\n"
"   -h              help menu (this screen)\n"
"   -i              interactive mode (doesn't do anything)\n"
"   -f              force mode (doesn't do anything)\n"
"   -r              recursive mode (doesn't do anything)\n"
"   -v[level]       set verbosity level (5 is default; doesn't do
anything)\n"
"   -o filename     set output filename (doesn't do anything)\n"
    );
}


/*********************************************************************
********
* MAIN
*/

int main(int argc, char* argv[])
{
    /* check arguments */
    while(1) {
        int c = getopt(argc, argv, "-ifrhv::o:");
        if(c == -1) break;

        switch(c) {
            case 'i': flags |= FLAG_INTERACT; break;
            case 'f': flags |= FLAG_FORCE; break;
            case 'r': flags |= FLAG_RECURSIVE; break;

            case 'h': help(); exit(0);

            case 'v': verbose = arg_to_int(optarg, 0, 10, 5, "v");
break;
```

```
            case 'o': out_fname = optarg; break;
            case 1: in_fname = optarg; break;

            #ifdef DEBUG
            default:
                printf("Option '%c' (%d) with '%s'\n", c, c, optarg);
            #endif
        }
    }

    #ifdef DEBUG
        printf("optind at %d; argv[optind] = '%s'\n", optind,
argv[optind]);
    #endif

    /* print out what we got */
    if(flags & FLAG_INTERACT) printf("in interactive mode\n");
    else printf("not in interactive mode\n");
    if(flags & FLAG_FORCE) printf("in force mode\n");
    else printf("not in force mode\n");
    if(flags & FLAG_RECURSIVE) printf("in recursive mode\n");
    else printf("not in recursive mode\n");
    printf("verbosity level: %d\n", verbose);
    if(in_fname) printf("input filename: %s\n", in_fname);
    else printf("no input filename\n");
    if(out_fname) printf("output filename: %s\n", out_fname);
    else printf("no output filename\n");

    return 0;
}


/* vim:ts=3
*/
```