6.1.5. Напишите функцию рекурсивного обхода дерева подкаталогов и печати имен всех файлов в нем. Ключ U42 означает файловую систему с длинными именами файлов (BSD 4.2).

```
/*#!/bin/cc -DFIND -DU42 -DMATCHONLY treemk.c match.c -o tree -lx
 * Обход поддерева каталогов (по мотивам Керниган & Ритчи).
 *              Ключи компиляции:
 * BSD-4.2 BSD-4.3                      -DU42
 * XENIX с канонической файл.сист.      ничего
 * XENIX с библиотекой  -lx             -DU42
 *      программа поиска файлов                         -DFIND
 *      программа рекурсивного удаления                 -DRM_REC
 *      программа подсчета используемого места на диске  БЕЗ_КЛЮЧА
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>         /* для MAXPATHLEN */

#if defined(M_XENIX) && defined(U42)
# include <sys/ndir.h>  /* XENIX + U42 эмуляция */
#else
# include <dirent.h>
# define stat(f,s) lstat(f,s)  /* не проходить по символьным ссылкам */
# define d_namlen d_reclen
#endif

/* проверка: каталог ли это */
#define  isdir(st) ((st.st_mode & S_IFMT) == S_IFDIR)
struct   stat st;               /* для сисвызова stat() */
char     buf[MAXPATHLEN+1];     /* буфер для имени файла */

#define FAILURE (-1)            /* код неудачи */
#define SUCCESS   1             /* код успеха  */
#define WARNING   0             /* нефатальная ошибка */
/* Сообщения об ошибках во время обхода дерева: */
#ifndef ERR_CANT_READ
# define ERR_CANT_READ(name)  \
         fprintf( stderr, "\tНе могу читать \"%s\"\n", name), WARNING
# define ERR_NAME_TOO_LONG()  \
         fprintf( stderr, "\tСлишком длинное полное имя\n" ), WARNING
#endif

/* Прототипы для предварительного объявления функций. */
extern char *strrchr(char *, char);
int directory (char *name, int level,
    int (*enter)(char *full, int level, struct stat *st),
    int (*leave)(char *full, int level),
    int (*touch)(char *full, int level, struct stat *st));
/* Функции-обработчики enter, leave, touch должны
 * возвращать (-1) для прерывания просмотра дерева,
 * либо значение >= 0 для продолжения. */
/* Обойти дерево с корнем в rootdir */
int walktree (
    char *rootdir,      /* корень дерева */
    int (*enter)(char *full, int level, struct stat *st),
    int (*leave)(char *full, int level),
    int (*touch)(char *full, int level, struct stat *st)
){
    /* проверка корректности корня */
    if( stat(rootdir, &st) < 0 || !isdir(st)){
        fprintf( stderr, "\tПлохой корень дерева \"%s\"\n", rootdir );
        return   FAILURE;  /* неудача */
    }
    strcpy     (buf, rootdir);
    return act (buf, 0, enter, leave, touch);
}

/* Оценка файла с именем name.
 */
int act (char *name, int level,
```

```
        int (*enter)(char *full, int level, struct stat *st),
        int (*leave)(char *full, int level),
        int (*touch)(char *full, int level, struct stat *st))
    {
        if (stat (name, &st) < 0)
            return WARNING; /* ошибка, но не фатальная      */
        if(isdir(st)){      /* позвать обработчик каталогов */
           if(enter)
              if( enter(name, level, &st) == FAILURE ) return FAILURE;
           return directory (name, level+1, enter, leave, touch);

        } else {            /* позвать обработчик файлов    */
           if(touch) return touch (name, level, &st);
           else      return SUCCESS;
        }
    }

    /* Обработать каталог: прочитать его и найти подкаталоги */
    int directory (char *name, int level,
        int (*enter)(char *full, int level, struct stat *st),
        int (*leave)(char *full, int level),
        int (*touch)(char *full, int level, struct stat *st))
    {
    #ifndef U42
        struct direct   dirbuf;
        int        fd;
    #else
        register struct dirent *dirbuf;
        DIR    *fd;
        extern DIR *opendir();
    #endif
        char   *nbp, *tail, *nep;
        int     i, retcode = SUCCESS;

    #ifndef U42
        if ((fd = open (name, 0)) < 0) {
    #else
        if ((fd = opendir (name)) == NULL) {
    #endif
            return ERR_CANT_READ(name);
        }

        tail = nbp = name + strlen (name);  /* указатель на закрывающий \0 */
        if( strcmp( name, "/" ))  /* если не "/" */
            *nbp++ = '/';
        *nbp = '\0';

    #ifndef U42
        if (nbp + DIRSIZ + 2 >= name + MAXPATHLEN) {
            *tail = '\0';
            return ERR_NAME_TOO_LONG();
        }
    #endif

    #ifndef U42
        while (read(fd, (char *) &dirbuf, sizeof(dirbuf)) == sizeof(dirbuf)){
            if (dirbuf.d_ino == 0)  /* стертый файл */
                continue;
            if (strcmp (dirbuf.d_name, "." ) == 0  ||
                strcmp (dirbuf.d_name, "..") == 0)  /* не интересуют */
                continue;
            for (i = 0, nep = nbp; i < DIRSIZ; i++)
                *nep++ = dirbuf.d_name[i];

    # else /*U42*/
        while ((dirbuf = readdir (fd)) != NULL ) {
            if (dirbuf->d_ino == 0)
                continue;
            if (strcmp (dirbuf->d_name, "." ) == 0  ||
                strcmp (dirbuf->d_name, "..") == 0)
                continue;
            for (i = 0, nep = nbp; i < dirbuf->d_namlen ; i++)
                *nep++ = dirbuf->d_name[i];
    #endif /*U42*/
            *nep = '\0';
            if( act(name, level,  enter, leave, touch) == FAILURE) {
                retcode = FAILURE; break;                          }
        }

    #ifndef U42
        close (fd);
    #else
        closedir(fd);
    #endif
```

```
    *tail = '\0';       /* восстановить старое name */

    if(retcode != FAILURE   &&   leave)
       if( leave(name, level) == FAILURE) retcode = FAILURE;
    return retcode;
}

/* ------------------------------------------------------------ */
/* Disk Usage -- Оценка места, занимаемого файлами поддерева    */
/* ------------------------------------------------------------ */
/* Пересчет байтов в килобайты */
#define KB(s)  (((s)/1024L) + ((s)%1024L ? 1L:0L))
/* или #define KB(s)   (((s) + 1024L - 1) / 1024L)  */
long size;                      /* общий размер     */
long nfiles;                    /* всего файлов     */
long ndirs;                     /* из них каталогов */
#define WARNING_LIMIT 150L      /* подозрительно большой файл */

static int du_touch (char *name, int level, struct stat *st){
     long sz;
     size += (sz = KB(st->st_size));  /* размер файла в Кб. */
     nfiles++;
#ifndef TREEONLY
     if( sz >= WARNING_LIMIT )
        fprintf(stderr,"\tВнимание! \"%s\" очень большой: %ld Кб.\n",
                                    name,               sz);
#endif /*TREEONLY*/
     return SUCCESS;
}
static int du_enter (char *name, int level, struct stat *st){
#ifndef TREEONLY
     fprintf( stderr, "Каталог \"%s\"\n", name );
#endif
     size += KB(st->st_size);  /* размер каталога в Кб. */
     nfiles++; ++ndirs; return SUCCESS;
}
long du (char *name){
     size = nfiles = ndirs = 0L;
     walktree(name, du_enter, NULL, du_touch );
     return size;
}

/* ------------------------------------------------------------ */
/* Рекурсивное удаление файлов и каталогов                      */
/* ------------------------------------------------------------ */
int  deleted;    /* сколько файлов и каталогов удалено */
static int recrm_dir (char *name, int level){
     if( rmdir(name) >= 0){ deleted++; return SUCCESS; }
     fprintf(stderr, "Не могу rmdir '%s'\n", name); return WARNING;
}

static int recrm_file(char *name, int level, struct stat *st){
     if( unlink(name) >= 0){ deleted++; return SUCCESS; }
     fprintf(stderr, "Не могу rm    '%s'\n", name); return WARNING;
}
int recrmdir(char *name){
    int ok_code; deleted = 0;
    ok_code = walktree(name, NULL, recrm_dir, recrm_file);
    printf("Удалено %d файлов и каталогов в %s\n", deleted, name);
    return ok_code;
}

/* ------------------------------------------------------------ */
/* Поиск файлов с подходящим именем (по шаблону имени)          */
/* ------------------------------------------------------------ */
char *find_PATTERN;
static int find_check(char *fullname, int level, struct stat *st){
    char *basename = strrchr(fullname, '/');
    if(basename) basename++;
    else         basename = fullname;
    if( match(basename, find_PATTERN))
        printf("Level#%02d %s\n", level, fullname);
    if( !strcmp( basename, "core")){
        printf("Найден дамп %s, поиск прекращен.\n", fullname);
        return FAILURE;
    }
    return SUCCESS;
}
void find (char *root, char *pattern){
     find_PATTERN = pattern;
     walktree(root, find_check, NULL, find_check);
}

/* ------------------------------------------------------------ */
```

```
#ifndef TREEONLY
void main(int argc, char *argv[]){
#ifdef FIND
     if(argc != 3){ fprintf(stderr, "Arg count\n"); exit(1); }
     find(argv[1], argv[2]);
#else
# ifdef RM_REC
     for(argv++; *argv; argv++)
         recrmdir(*argv);
# else
     du( argc == 1 ? "." : argv[1] );
     printf( "%ld килобайт в %ld файлах.\n", size, nfiles );
     printf( "%ld каталогов.\n", ndirs );
# endif
#endif
     exit(0);
}
#endif /*TREEONLY*/
```

6.1.6. Используя предыдущий алгоритм, напишите программу рекурсивного копирования поддерева каталогов в другое место. Для создания новых каталогов используйте системный вызов

```
mkdir(имя_каталога, коды_доступа);
```

6.1.7. Используя тот же алгоритм, напишите программу удаления каталога, которая удаляет все файлы в нем и, рекурсивно, все его подкаталоги. Таким образом, удаляется дерево каталогов. В *UNIX* подобную операцию выполняет команда

```
rm -r имя_каталога_корня_дерева
```

6.1.8. Используя все тот же алгоритм обхода, напишите аналог команды *find*, который будет позволять:

- находить все файлы, чьи имена удовлетворяют заданному шаблону (используйте функцию *match*() из главы "Текстовая обработка");
- находить все выполняемые файлы: обычные файлы *S_IFREG*, у которых

```
(st.st_mode & 0111) != 0
```

Как уже ясно, следует пользоваться вызовом *stat* для проверки каждого файла.

6.2. Время в UNIX.

6.2.1. Напишите функцию, переводящую год, месяц, день, часы, минуты и секунды в число секунд, прошедшее до указанного момента с 00 часов 00 минут 00 секунд 1 Января 1970 года. Внимание: результат должен иметь тип *long* (точнее *time_t*).

Эта функция облегчит вам сравнение двух моментов времени, заданных в общепринятом "человеческом" формате, поскольку сравнить два *long* числа гораздо проще, чем сравнивать по очереди годы, затем, если они равны – месяцы, если месяцы равны – даты, и.т.д.; а также облегчит измерение интервала между двумя событиями – он вычисляется просто как разность двух чисел. В системе *UNIX* время обрабатывается и хранится именно в виде числа секунд; в частности текущее астрономическое время можно узнать системным вызовом

```
#include <sys/types.h>
#include <time.h>
time_t t = time(NULL);  /* time(&t); */
```

Функция

```
struct tm *tm = localtime( &t );
```

разлагает число секунд на отдельные составляющие, содержащиеся в int-полях структуры:

```
tm_year  год           (надо прибавлять 1900)
tm_yday  день в году   0..365
tm_mon   номер месяца  0..11 (0 - Январь)
tm_mday  дата месяца   1..31
tm_wday  день недели   0..6  (0 - Воскресенье)
tm_hour  часы          0..23
tm_min   минуты        0..59
tm_sec   секунды       0..59
```

Номера месяца и дня недели начинаются с нуля, чтобы вы могли использовать их в качестве индексов:

```
char *months[] = { "Январь", "Февраль", ..., "Декабрь" };
printf( "%s\n", months[ tm->tm_mon ] );
```

Пример использования этих функций есть в приложении. Установить время в системе может суперпользователь вызовом

```
stime(&t);
```

6.2.2. Напишите функцию печати текущего времени в формате ЧЧ:ММ:СС ДД-МЕС-ГГ. Используйте системный вызов *time*() и функцию *localtime*().

Существует стандартная функция *ctime*(), которая печатает время в формате:

```
/* Mon Mar 25 18:56:36 1991 */
#include <stdio.h>
#include <time.h>
main(){ /* команда date */
        time_t t = time(NULL);
        char *s  = ctime(&t);
        printf("%s", s);
}
```

Обратите внимание, что строка **s** уже содержит на конце символ '\n'.

6.2.3. Структура *stat*, заполняемая системным вызовом *stat*(), кроме прочих полей содержит поля типа *time_t* **st_ctime**, **st_mtime** и **st_atime** - время последнего изменения содержимого I-узла файла, время последнего изменения файла и время последнего доступа к файлу.

- Поле **st_ctime** изменяется (устанавливается равным текущему астрономическому времени) при применении к файлу вызовов *creat*, *chmod*, *chown*, *link*, *unlink*, *mknod*, *utime*[*], *write* (т.к. изменяется длина файла); Это поле следует рассматривать как время модификации прав доступа к файлу;
- **st_mtime** - *write*, *creat*, *mknod*, *utime*; Это поле следует рассматривать как время модификации содержимого файла (данных);
- **st_atime** - *read*, *creat*, *mknod*, *utime*; Это поле следует рассматривать как время чтения содержимого файла (данных).

Модифицируйте функцию *typeOf*(), чтобы она печатала еще и эти даты.

```
utime(имяФайла, NULL);
```

Он используется для взаимодействия с программой *make* - в команде *touch*. Изменить время можно только своему файлу.

6.2.4. Напишите аналог команды *ls -tm*, выдающей список имен файлов текущего каталога, отсортированный по убыванию поля **st_mtime**, то есть недавно модифицированные файлы выдаются первыми. Для каждого прочитанного из каталога имени надо сделать *stat*; имена файлов и времена следует сохранить в массиве структур, а затем отсортировать его.

6.2.5. Напишите аналогичную программу, сортирующую файлы в порядке возрастания их размера (**st_size**).

6.2.6. Напишите аналог команды *ls -l*, выдающий имена файлов каталога и их коды доступа в формате **rwxrw-r--**. Для получения кодов доступа используйте вызов *stat*

```
stat( имяФайла, &st);
кодыДоступа = st.st_mode & 0777;
```

Для изменения кодов доступа используется вызов

```
chmod(имя_файла, новые_коды);
```

Можно изменять коды доступа, соответствующие битовой маске

```
0777 | S_ISUID | S_ISGID | S_ISVTX
```

(смотри <**sys/stat.h**>). **Тип** файла (см. функцию *typeOf*) не может быть изменен. Изменить коды доступа к файлу может только его владелец.

Печатайте еще номер I-узла файла: поле **d_ino** каталога либо поле **st_ino** структуры *stat*.

6.2.7. Вот программа, которая каждые 2 секунды проверяет - не изменилось ли содержимое текущего каталога:

```
#include <sys/types.h>
#include <sys/stat.h>
extern char *ctime();
main(){
```

```
        time_t last; struct stat st;
        for( stat(".", &st), last=st.st_mtime; ; sleep(2)){
             stat(".", &st);
             if(last != st.st_mtime){
                last  = st.st_mtime;
    printf("Был создан или удален какой-то файл: %s",
                      ctime(&last));
             }
        }
    }
```

Модифицируйте ее, чтобы она сообщала **какое** имя (имена) было удалено или создано (для этого надо при запуске программы прочитать и запомнить содержимое каталога, а при обнаружении модификации – перечитать каталог и сравнить его с прежним содержимым).

6.2.8. Напишите по аналогии программу, которая выдает сообщение, если указанный вами файл был кем-то прочитан, записан или удален. Вам следует отслеживать изменение полей **st_atime**, **st_mtime** и значение *stat()* < 0 соответственно. Если файл удален – программа завершается.

6.2.9. Современные *UNIX*-машины имеют встроенные таймеры (как правило несколько) с довольно высоким разрешением. Некоторые из них могут использоваться как "будильники" с обратным отсчетом времени: в таймер загружается некоторое значение; таймер ведет обратный отсчет, уменьшая загруженный счетчик; как только это время истекает – посылается сигнал процессу, загрузившему таймер.

Вот как, к примеру, выглядит функция задержки в микросекундах (миллионных долях секунды). Примечание: эту функцию не следует использовать вперемежку с функциями *sleep* и *alarm* (смотри статью про них ниже, в главе про сигналы).

```
#include <sys/types.h>
#include <signal.h>
#include <sys/time.h>

void do_nothing() {}

/* Задержка на usec миллионных долей секунды (микросекунд) */
void usleep(unsigned int usec) {

        struct itimerval        new, old;
        /*  struct itimerval  содержит поля:
            struct timeval    it_interval;
            struct timeval    it_value;

            Где struct timeval содержит поля:
            long    tv_sec;    -- число целых секунд
            long    tv_usec;   -- число микросекунд
         */
        struct sigaction        new_vec, old_vec;

        if (usec == 0) return;

        /* Поле tv_sec  содержит число целых секунд.
           Поле tv_usec содержит число микросекунд.

           it_value    - это время, через которое В ПЕРВЫЙ раз
                         таймер "прозвонит",
                         то есть пошлет нашему процессу
                         сигнал SIGALRM.

                         Время, равное нулю, немедленно остановит таймер.

           it_interval - это интервал времени, который будет загружаться
                         в таймер после каждого "звонка"
                         (но не в первый раз).

                         Время, равное нулю, остановит таймер
                         после его первого "звонка".
         */
        new.it_interval.tv_sec  = 0;
        new.it_interval.tv_usec = 0;
        new.it_value.tv_sec  = usec / 1000000;
        new.it_value.tv_usec = usec % 1000000;

        /* Сохраняем прежнюю реакцию на сигнал SIGALRM в old_vec,
           заносим в качестве новой реакции do_nothing()
         */
        new_vec.sa_handler = do_nothing;
        sigemptyset(&new_vec.sa_mask);
        new_vec.sa_flags = 0;

        sighold(SIGALRM);
```

```
        sigaction(SIGALRM, &new_vec, &old_vec);

        /* Загрузка интервального таймера значением new, начало отсчета.
         * Прежнее значение спасти в old.
         * Вместо &old можно также NULL - не спасать.
         */
        setitimer(ITIMER_REAL, &new, &old);

        /* Ждать прихода сигнала SIGALRM */
        sigpause(SIGALRM);

        /* Восстановить реакцию на SIGALRM */
        sigaction(SIGALRM, &old_vec, (struct sigaction *) 0);
        sigrelse(SIGALRM);

        /* Восстановить прежние параметры таймера */
        setitimer(ITIMER_REAL, &old, (struct itimerval *) 0);
    }
```

* - Время модификации файла можно изменить на текущее астрономическое время и не производя записи в файл. Для этого используется вызов