Fall 2004 Handout 1

Syllabus (not in chronological order)

(1) C programming and data structures. Structures and arrays of structures. Build linked lists and trees out of dynamically allocated structures linked by pointers. Recursion. Hash tables.

(2) Unix utilities: **`gcc`** to compile and link a C program, **`nm`** and **`strip`** to display and remove a C program's symbol table or ''name table'', **`ar`** to create a library or ''archive'', **`time`** and **`size`** to measure the speed and static size of an executable, **`rcs`** to log changes to the source code, and **`make`** to run all the above commands.

(3) Unix system calls to get information about files and directories: **`opendir`**, **`readdir`**, **`closedir`**, **`stat`**.

(4) Unix system calls to spawn processes (i.e., run programs): **`fork`**, **`exec`**, **`wait`**. Parent and child. Give each process its own source of standard input and destination of standard output: **`dup2`**, **`close`**. Connect the processes with pipes: **`pipe`**.

(5) Unix system calls to communicate via sockets with programs running on other machines: **`socket`**, **`connect`**, **`bind`**, **`listen`**, **`accept`**, **`shutdown`**. Clients and servers (two more uses for **`fork`**'ing).

(6) Unix system calls to create and access blocks of shared memory: **`shmget`**, **`shmctl`**, **`shmat`**, **`shmdt`** (another use for pointers to structures).

(7) Unix system calls to synchronize processes using semaphores: **`semget`**, **`semctl`**, **`semop`**.

(8) Unix system calls for multi-threaded programs using Pthreads. Communication and synchronization between threads.

(9) Miscellaneous and/or optional. Unix system calls for signals, non-blocking i/o, etc. Call a Fortran function or subroutine from a C program. Objects in C++ vs. pointers to structures as function arguments in C.

Three big projects: build your own...

(1) **`ls -l`**. A file on the disk is a stationary target.

(2) **`sh`**. A process is a moving target.

(3) **`telnet`**. Requires communication with another machine.

Array of structures

```
Please type your weight: 150
On each planet, your weight would be
Mercury      40.50
Venus       127.50
Earth       150.00      etc.
```

—Source code on the Web at
`http://i5.nyu.edu/~mm64/x52.9232/src/planet_wo_arrays.c`

```
1 #include <stdio.h>   /* for printf, scanf */
2 #include <stdlib.h>  /* for exit, EXIT_SUCCESS, EXIT_FAILURE */
3
```

```
 4 int main()
 5 {
 6     double weight;
 7
 8     printf("Please type your weight: ");
 9     scanf("%lf", &weight);
10     printf("On each planet, your weight would be\n");
11
12     printf("%-11s %6.2f\n", "Mercury",  .27 * weight);
13     printf("%-11s %6.2f\n", "Venus",    .85 * weight);
14     printf("%-11s %6.2f\n", "Earth",   1.00 * weight);
15     printf("%-11s %6.2f\n", "Mars",     .38 * weight);
16     printf("%-11s %6.2f\n", "Jupiter", 2.33 * weight);
17     printf("%-11s %6.2f\n", "Saturn",   .92 * weight);
18     printf("%-11s %6.2f\n", "Uranus",   .85 * weight);
19     printf("%-11s %6.2f\n", "Neptune", 1.12 * weight);
20     printf("%-11s %6.2f\n", "Pluto",    .44 * weight);
21
22     return EXIT_SUCCESS;
23 }
```

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9232/src/planet2arrays.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 char *name[] = {
 5     "Mercury",
 6     "Venus",
 7     "Earth",
 8     "Mars",
 9     "Jupiter",
10     "Saturn",
11     "Uranus",
12     "Neptune",
13     "Pluto"
14 };
15 #define N (sizeof name / sizeof name[0])  /* the number of planets */
16
17 double factor[] = {  /* gravity compared to earth's */
18      .27,        /* Mercury */
19      .85,        /* Venus */
20     1.00,        /* Earth (by definition) */
21      .38,        /* Mars */
22     2.33,        /* Jupiter */
23      .92,        /* Saturn */
24      .85,        /* Uranus */
25     1.12,        /* Neptune */
26      .44         /* Pluto */
27 };
28
29 int main()
30 {
31     double weight;
32     int i;
```



```
33      char **pn;
34      double *pf;
35
36      printf("Please type your weight: ");
37      scanf("%lf", &weight);
38      printf("On each planet, your weight would be\n");
39
40      for (i = 0; i < N; ++i) {
41          printf("%-11s %6.2f\n", name[i], factor[i] * weight);
42      }
43
44      for (pn = name, pf = factor; pn < name + N; ++pn, ++pf) {
45          printf("%-11s %6.2f\n", *pn, *pf * weight);
46      }
47
48      return EXIT_SUCCESS;
49  }
```

—Source code on the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/planet_array_of_structures.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  int main()
 5  {
 6      typedef struct {
 7          char *name;  /* name of the planet */
 8          double factor;   /* how many times stronger its gravity is than earth's */
 9      } planet_t;
10
11      planet_t a[] = {
12          {"Mercury",       .27},
13          {"Venus",         .85},
14          {"Earth",        1.00},
15          {"Mars",          .38},
16          {"Jupiter",      2.33},
17          {"Saturn",        .92},
18          {"Uranus",        .85},
19          {"Neptune",      1.12},
20          {"Pluto",         .44}
21      };
22  #define N (sizeof a / sizeof a[0])        /* the number of planets */
23
24      double weight;
25      int i;
26      planet_t *p;
27
28      printf("Please type your weight: ");
29      scanf("%lf", &weight);
30      printf("On each planet, your weight would be\n");
31
32      for (i = 0; i < N; ++i) {
33          printf("%-11s %6.2f\n", a[i].name, weight * a[i].factor);
34      }
```

```
35
36      for (p = a; p < a + N; ++p) {
37          printf("%-11s %6.2f\n", p->name, weight * p->factor);
38      }
39
40      return EXIT_SUCCESS;
41  }
```

Here's another way to stop the **for** loop:

—Source code on the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/planet_array_of_structures2.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  int main()
 5  {
 6      typedef struct {
 7          char *name;  /* name of the planet */
 8          double factor;   /* how many times stronger its gravity is than earth's */
 9      } planet_t;
10
11      planet_t a[] = {
12          {"Mercury",  .27},
13          {"Venus",    .85},
14          {"Earth",   1.00},
15          {"Mars",     .38},
16          {"Jupiter", 2.33},
17          {"Saturn",   .92},
18          {"Uranus",   .85},
19          {"Neptune", 1.12},
20          {"Pluto",    .44},
21
22          {NULL,      0.00}    /* dummy to terminate the array */
23      };
24
25      double weight;
26      int i;
27      planet_t *p;
28
29      printf("Please type your weight: ");
30      scanf("%lf", &weight);
31      printf("On each planet, your weight would be\n");
32
33      for (i = 0; a[i].name != NULL; ++i) {
34          printf("%-11s %6.2f\n", a[i].name, weight * a[i].factor);
35      }
36
37      for (p = a; p->name != NULL; ++p) {
38          printf("%-11s %6.2f\n", p->name, weight * p->factor);
39      }
40
41      return EXIT_SUCCESS;
42  }
```

How not to program in any language

```
 1 int BigFieldJustChanged(char *);    /* return 0 for false, non-zero for true */
 2
 3     char CELL[3][3];                /* tic-tac-toe board */
 4     char WINNER[2];                 /* an empty string */
 5
 6     if (BigFieldJustChanged(&CELL[0][0])) {
 7         if (CELL[0][0] == 'X' && CELL[0][1] == 'X' && CELL[0][2] == 'X' ||
 8             CELL[0][0] == 'X' && CELL[1][0] == 'X' && CELL[2][0] == 'X' ||
 9             CELL[0][0] == 'X' && CELL[1][1] == 'X' && CELL[2][2] == 'X') {
10             strcpy(WINNER, "X");
11         }
12
13         if (CELL[0][0] == 'O' && CELL[0][1] == 'O' && CELL[0][2] == 'O' ||
14             CELL[0][0] == 'O' && CELL[1][0] == 'O' && CELL[2][0] == 'O' ||
15             CELL[0][0] == 'O' && CELL[1][1] == 'O' && CELL[2][2] == 'O') {
16             strcpy(WINNER, "O");
17         }
18     }
19
20     if (BigFieldJustChanged(&CELL[1][0])) {
21         if (CELL[1][0] == 'X' && CELL[1][1] == 'X' && CELL[1][2] == 'X' ||
22             CELL[0][0] == 'X' && CELL[1][0] == 'X' && CELL[2][0] == 'X') {
23             strcpy(WINNER, "X");
24         }
25
26         if (CELL[1][0] == 'O' && CELL[1][1] == 'O' && CELL[1][2] == 'O' ||
27             CELL[0][0] == 'O' && CELL[1][0] == 'O' && CELL[2][0] == 'O') {
28             strcpy(WINNER, "O");
29         }
30     }
31
32     if (BigFieldJustChanged(&CELL[2][0])) {
33         if (CELL[2][0] == 'X' && CELL[2][1] == 'X' && CELL[2][2] == 'X' ||
34             CELL[2][0] == 'X' && CELL[1][0] == 'X' && CELL[0][0] == 'X' ||
35             CELL[2][0] == 'X' && CELL[1][1] == 'X' && CELL[0][2] == 'X') {
36             strcpy(WINNER, "X");
37         }
38
39         if (CELL[2][0] == 'O' && CELL[2][1] == 'O' && CELL[2][2] == 'O' ||
40             CELL[2][0] == 'O' && CELL[1][0] == 'O' && CELL[0][0] == 'O' ||
41             CELL[2][0] == 'O' && CELL[1][1] == 'O' && CELL[0][2] == 'O') {
42             strcpy(WINNER, "O");
43         }
44     }
45
46     if (BigFieldJustChanged(&CELL[0][1])) {
47         if (CELL[0][1] == 'X' && CELL[1][1] == 'X' && CELL[2][1] == 'X' ||
48             CELL[0][1] == 'X' && CELL[0][0] == 'X' && CELL[0][2] == 'X') {
49             strcpy(WINNER, "X");
50         }
51
52         if (CELL[0][1] == 'O' && CELL[1][1] == 'O' && CELL[2][1] == 'O' ||
53             CELL[0][1] == 'O' && CELL[0][0] == 'O' && CELL[0][2] == 'O') {
```



```
54              strcpy(WINNER, "O");
55          }
56      }
57
58      if (BigFieldJustChanged(&CELL[1][1])) {
59          if (CELL[1][1] == 'X' && CELL[2][1] == 'X' && CELL[0][1] == 'X' ||
60              CELL[1][1] == 'X' && CELL[1][0] == 'X' && CELL[1][2] == 'X' ||
61              CELL[1][1] == 'X' && CELL[0][0] == 'X' && CELL[2][2] == 'X' ||
62              CELL[1][1] == 'X' && CELL[0][2] == 'X' && CELL[2][0] == 'X') {
63              strcpy(WINNER, "X");
64          }
65
66          if (CELL[1][1] == 'O' && CELL[2][1] == 'O' && CELL[0][1] == 'O' ||
67              CELL[1][1] == 'O' && CELL[1][0] == 'O' && CELL[1][2] == 'O' ||
68              CELL[1][1] == 'O' && CELL[0][0] == 'O' && CELL[2][2] == 'O' ||
69              CELL[1][1] == 'O' && CELL[0][2] == 'O' && CELL[2][0] == 'O') {
70              strcpy(WINNER, "O");
71          }
72      }
73
74      if (BigFieldJustChanged(&CELL[2][1])) {
75          if (CELL[2][1] == 'X' && CELL[2][0] == 'X' && CELL[2][2] == 'X' ||
76              CELL[2][1] == 'X' && CELL[0][1] == 'X' && CELL[1][1] == 'X') {
77              strcpy(WINNER, "X");
78          }
79
80          if (CELL[2][1] == 'O' && CELL[2][0] == 'O' && CELL[2][2] == 'O' ||
81              CELL[2][1] == 'O' && CELL[0][1] == 'O' && CELL[1][1] == 'O') {
82              strcpy(WINNER, "O");
83          }
84      }
85
86      if (BigFieldJustChanged(&CELL[0][2])) {
87          if (CELL[0][2] == 'X' && CELL[1][2] == 'X' && CELL[2][2] == 'X' ||
88              CELL[0][2] == 'X' && CELL[0][1] == 'X' && CELL[0][0] == 'X' ||
89              CELL[0][2] == 'X' && CELL[1][1] == 'X' && CELL[2][0] == 'X') {
90              strcpy(WINNER, "X");
91          }
92
93          if (CELL[0][2] == 'O' && CELL[1][2] == 'O' && CELL[2][2] == 'O' ||
94              CELL[0][2] == 'O' && CELL[0][1] == 'O' && CELL[0][0] == 'O' ||
95              CELL[0][2] == 'O' && CELL[1][1] == 'O' && CELL[2][0] == 'O') {
96              strcpy(WINNER, "O");
97          }
98      }
99
100     if (BigFieldJustChanged(&CELL[1][2])) {
101         if (CELL[1][2] == 'X' && CELL[0][2] == 'X' && CELL[2][2] == 'X' ||
102             CELL[1][2] == 'X' && CELL[1][1] == 'X' && CELL[1][0] == 'X') {
103             strcpy(WINNER, "X");
104         }
105
106         if (CELL[1][2] == 'O' && CELL[0][2] == 'O' && CELL[2][2] == 'O' ||
107             CELL[1][2] == 'O' && CELL[1][1] == 'O' && CELL[1][0] == 'O') {
```

```
108             strcpy(WINNER, "O");
109         }
110     }
111
112     if (BigFieldJustChanged(&CELL[2][2])) {
113         if (CELL[2][2] == 'X' && CELL[1][2] == 'X' && CELL[0][2] == 'X' ||
114             CELL[2][2] == 'X' && CELL[2][1] == 'X' && CELL[2][0] == 'X' ||
115             CELL[2][2] == 'X' && CELL[1][1] == 'X' && CELL[0][0] == 'X') {
116             strcpy(WINNER, "X");
117         }
118
119         if (CELL[2][2] == 'O' && CELL[1][2] == 'O' && CELL[0][2] == 'O' ||
120             CELL[2][2] == 'O' && CELL[2][1] == 'O' && CELL[2][0] == 'O' ||
121             CELL[2][2] == 'O' && CELL[1][1] == 'O' && CELL[0][0] == 'O') {
122             strcpy(WINNER, "O");
123         }
124     }
```

Invent two new data types: cell_t and line_t

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define N 3             /* number of rows == number of columns */
 5 char CELL[N][N];         /* tic-tac-toe board, inherited from previous program. */
 6
 7 /* Each cell has a row number and a column number. */
 8 typedef struct {
 9     int row;
10     int col;
11 } cell_t;
12
13 /* A line is N cells in a straight line. */
14 typedef cell_t line_t[N];
15
16 /* Here are the lines on a tic-tac-toe board: */
17
18 const line_t line[] = {
19     {{0, 0}, {0, 1}, {0, 2}},       /* top row */
20     {{1, 0}, {1, 1}, {1, 2}},       /* middle row */
21     {{2, 0}, {2, 1}, {2, 2}},       /* bottom row */
22
23     {{0, 0}, {1, 0}, {2, 0}},       /* left column */
24     {{0, 1}, {1, 1}, {2, 1}},       /* middle column */
25     {{0, 2}, {1, 2}, {2, 2}},       /* right column */
26
27     {{0, 0}, {1, 1}, {2, 2}},       /* main diagonal (upper left to lower right) */
28     {{0, 2}, {1, 1}, {2, 0}}        /* other diagonal (upper right to lower left) */
29 };
30 #define NLINES (sizeof line / sizeof line[0])
31
32 int BigFieldJustChanged(char *cell);  /* inherited from previous program */
33
34 cell_t which_cell_just_changed();
```

```
35 int contains(const line_t line, cell_t cell);
36 char all_the_same(const line_t line);
37
38 int main(int argc, char **argv)
39 {
40     int i;  /* index into line[] array (better name than l) */
41     char c;
42     char WINNER[] = "?"; /* inherited from previous program */
43
44
45     cell_t cell = which_cell_just_changed();
46     if (cell.row == -1 || cell.col == -1) {
47         fprintf(stderr, %s: no cell changed.\n", argv[0]);
48         return EXIT_FAILURE;
49     }
50
51     for (i = 0; i < NLINES; ++i) {
52         if (contains(line[i], cell) && (c = all_the_same(line[i])) != '\0') {
53             WINNER[0] = c;
54             printf("The winner is %s\n", WINNER);
55             return EXIT_SUCCESS;
56         }
57     }
58
59     return EXIT_FAILURE;
60 }
61
62 /*
63 Return the cell_t that just changed.  This function does the work of the widely
64 scattered lines 6, 20, 32, 46, 58, 74, 89, 100, 112 of the original program.
65 */
66
67 cell_t which_cell_just_changed()
68 {
69     cell_t not_found = {-1, -1};
70
71     for (row = 0; row < N; ++row) {
72         for (col = 0; col < N; ++col) {
73             if (BigFieldJustChanged(&CELL[row][col])) {
74                 cell_t cell = {row, col};
75                 return cell;
76             }
77         }
78     }
79
80     return not_found;
81 }
82
83 /*
84 If the line contains the specified cell, return 1.  Otherwise, return 0.
85 */
86
87 int contains(const line_t line, cell_t cell)
88 {
```



```
 89      int i;
 90
 91      for (i = 0; i < N; ++i) {
 92          if (line[i].row == cell.row && line[i].col == cell.col) {
 93              return 1;
 94          }
 95      }
 96
 97      return 0;
 98  }
 99
100  /*
101  If all N characters in the line are the same, return the character.
102  Otherwise, return '\0'.
103  */
104
105  char all_the_same(const line_t line)
106  {
107      const char c = CELL[line[0].row][line[0].col];    /* first char in the line */
108      int i;
109
110      /* all the remaining char's */
111      for (i = 1; i < N; ++i) {
112          if (c != CELL[line[i].row][line[i].col]) {
113              return '\0';
114          }
115      }
116
117      return c;
118  }
```

Do it with a Perl regular expression

http://i5.nyu.edu/~mm64/x52.9544/src/winner

```
#!/bin/perl

$_ = <STDIN>;
chomp($_);  #Remove the trailing newline from $_.

if (length($_) != 9 || $_ =~ /[^XO ]/) {
    die "$0: Input line must be nine X's, O's, or blanks.";
}

#Insert a dash after the 3rd character and after the 6th character.
#For example, OOXXXOOXX becomes OOX-XXO-OXX.

$_ =~ s/(...)(...)(...)/\1-\2-\3/;

if (
    $_ =~ /([XO])\1\1/   ||      #any row
    $_ =~ /([XO])...\1...\1/ ||     #any column
    $_ =~ /([XO])..-.\1.-..\1/ ||   #the main diagonal
    $_ =~ /([XO])-.\1.-\1/) {       #the other diagonal

    print "$1 is a winner.\n";
    exit 0;
}

print "No one has won yet.\n";
exit 1;
```

Compile a C or C++ program

Compile a C program with **cc** or **gcc**; compile a C++ program with **cxx** or **g++**. Put your executable files (e.g., **prog**) in the **bin** subdirectory of your home directory:

```
1$ echo $PATH
2$ echo $PATH | tr : '\012' | more

3$ gcc -o ~/bin/prog prog.c              minus lowercase o to create an executable file named prog.
4$ gcc -o ~/bin/prog prog.c 2> errors    ksh(1) p. 18 for 2>
5$ more errors

6$ ls -l ~/bin/prog                      The execute bit will be turned on.
7$ prog                                  Execute prog.
```

Dbx tells what line your C program dumped core at.

```
1$ prog                                  prog dumps core.
2$ ls -l core                            Don't cat the core file.
```

Recompile your program with the **-g** option:


```
3$ gcc -g -o ~/bin/prog prog.c
4$ prog                          Run prog again; it will dump core again.
5$ ls -l core
6$ dbx ~/bin/prog core
(dbx) where                      main at bottom; most recent function at top
(dbx) quit
7$ rm core
```

Discover why a system called failed: Curry pp. 54−55; KP pp. 206−207; K&R p. 248

The error checking obscures the code, which is why they invented exceptions in C++:

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/error.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <errno.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     FILE *fp = fopen("outfile", "w");
 8     if (fp == NULL) {
 9         fprintf(stderr, "%s: fopen error number %d\n", argv[0], errno);
10         return EXIT_FAILURE;
11     }
12
13     if (fprintf(fp, "hello\n") != 6) {
14         fprintf(stderr, "%s: fprintf error number %d\n", argv[0], errno);
15         return EXIT_FAILURE;
16     }
17
18     if (fclose(fp) != 0) {
19         fprintf(stderr, "%s: fclose error number %d\n", argv[0], errno);
20         return EXIT_FAILURE;
21     }
22
23     return EXIT_SUCCESS;
24 }
```

The value of **errno**(3c) is one of the macros in **/usr/include/sys/errno.h**, which is **#include**'d by the file **/usr/include/errno.h**:

```
1 /* Excerpt from the file /usr/include/errno.h */
2 extern int errno;

3 /* Excerpts from the file /usr/include/sys/errno.h */
4 #define  EACCES   13   /* Permission denied         */
5 #define  EINVAL   22   /* Invalid argument          */
6 #define  ENFILE   23   /* File table overflow           */
7 #define  EMFILE   24   /* Too many open files           */
```

fopen(3c) lists all the reasons why **opendir** could fail. **intro**(2) and **/usr/include/sys/errno.h** list all the reasons why *any* Unix system call could fail—grim but fascinating reading.

The variable **errno** is in the object file **sparc_data.o** in the C Standard Library **/usr/lib/libc.a**:


```
1$ nm -Ap /usr/lib/libc.a | awk '$4 ~ /errno$/ && $3 ~ /[^NU]/'
/usr/lib/libc.a[errno.o]: 0000000000 T ___errno
/usr/lib/libc.a[sparc_data.o]: 0000000004 D errno
```

http://i5.nyu.edu/~mm64/x52.9544/src/error

```
#!/bin/ksh
#Here's how a shellscript can get its hands on the value of errno.

chmod a-w .
date > outfile
echo errno == $ERRNO
exit 0
```

```
2$ ./error
./error: /home1/m/mm64/outfile: cannot create
errno == 2
```

```
1 /* Excerpt from /usr/include/sys/errno.h:
2 #define  ENOENT   2   /* No such file or directory     */
```

▼ Homework 1.1: read the list of kernel error messages (not to be handed in)

Print and read **/usr/include/sys/errno.h** and **intro**(2). At Bloomberg,

```
1$ man -T lp 2 intro | lp -d prog_67
```

▲

Print an error message instead of an error number K&R p. 250

Instead of printing code number 13 for **EACCES**,

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <errno.h>
 4 #include <string.h>      /* for strerror */
 5
 6 int main(int argc, char **argv)
 7 {
 8     FILE *fp = fopen("outfile", "w");
 9
10     if (fp == NULL) {
11         fprintf(stderr, "%s: code %d means %s\n",
12             argv[0], errno, strerror(errno));
13         return EXIT_FAILURE;
14     }
```

```
1$ chmod a-w .                    Don't let anyone put a new file into the current directory.
2$ prog
prog: code 13 means Permission denied
```

▼ Homework 1.2: print every system call error message

Write a **for** loop that will call **strerror** and print its argument and return value. Right-justify the argument with **printf %3d**. Use the numbers 0, 1, 2, etc. as arguments of **strerror**. Keep going until you get a **NULL** return value. (In other versions of Unix, you keep going until the argument is

sys_nerr.)

Do not **#include errno.h** and do not use the variable **errno**: create your own variable to be the loop counter. Return exit status **EXIT_SUCCESS**.

Do not use **perror** in this assignment: the only functions you call will be **printf** and **strerror**, Look them up in Appendix B of K&R to find the **.h** files you must **#include**. You do not need to **#include errno.h** in order to call **strerror**.

Hand in the program and its output.

```
  0 Error 0
  1 Not owner
  2 No such file or directory
  3 No such process
  4 Interrupted system call
```

etc.

```
148 No route to host
149 Operation already in progress
150 Operation now in progress
151 Stale NFS file handle
```

Another possibility: use the above **for** loop to find the maximum legal value **n** for the argument of **strerror** but don't print anything yet. Then **sleep** for three seconds and pass a random number in the range 1 to **n** inclusive to **strerror**. Print **argv[0]**, a colon, a blank, the return value of **strerror**, and a **\n** to **stderr**, and **return** from **main** **exit** with exit status **EXIT_FAILURE**.

To make **rand** give you a different random number each time you run the program, call **srand** before **rand**:

```
 1 #include <stdlib.h>     /* for srand and rand */
 2 #include <time.h>       /* for time */
 3
 4 int main()
 5 {
 6     /* Seed the random number generator: */
 7     srand(time(NULL));
 8
 9     /* Now you can call rand. */
```

▲

perror: K&R p. 248

In all your homework, write an **if** testing the return value of every system call. If the value indicates failure, pass the name of the program (i.e., **argv[0]**) to **perror** and **exit** with a non-zero exit status:

```
 1 /* A better way to do the above example. */
 2 #include <stdio.h>       /* don't need string.h and errno.h to call perror */
 3 #include <stdlib.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     FILE *fp = fopen("outfile", "w");
 8     if (fp == NULL) {
 9         perror(argv[0]);
10         return EXIT_FAILURE;
11     }
```

```
1$ chmod a-w .                    Don't let anyone put a new file into the current directory.
2$ prog
prog: Permission denied
```

Why we have both perror and strerror

```
 1     FILE *fp = fopen("outfile", "w");
 2     if (fp == NULL) {
 3         perror(argv[0]);
 4         return EXIT_FAILURE;
 5     }

 6     FILE *fp1;
 7     FILE *fp2;
 8
 9     fp1 = fopen("outfile1", "w");
10     if (fp1 == NULL) {
11         fprintf(stderr, %s: outfile1: %s\n", argv[0], strerror(errno));
12         return EXIT_FAILURE;
13     }
14
15     fp2 = fopen("outfile2", "w");
16     if (fp2 == NULL) {
17         fprintf(stderr, %s: outfile2: %s\n", argv[0], strerror(errno));
18         return EXIT_FAILURE;
19     }
```

Print an error message in Perl

```
1$ prog1 && prog2
2$ prog1 || prog2
```

In Perl, a variable name *always* starts with a dollar sign: it's more consistent than the shell language. For example, **$0** is the name of the perlscript. As in the shell language, you can use variables within "double quotes" but not within 'single quotes':

```
3$ echo "Oh give me a $HOME"          examples in the shell language
Oh give me a /home1/a/abc1234

4$ echo 'Oh give me a $HOME'
Oh give me a $HOME
```

Outside of double quotes, the variable **$!** is the integer value of **errno**. But inside of double quotes, the variable **$!** is the error message that **strerror** would return.

http://i5.nyu.edu/~mm64/x52.9544/src/open

```
#!/bin/perl

if (!defined open(OUTFILE, ">outfile")) {
    die "$0: $!";
}

open(OUTFILE, ">outfile") || die "$0: $!";    #Requires parentheses.
open OUTFILE, ">/outfile" or die "$0: $!";    #Doesn't require parentheses.
die "$0: $!" unless open(OUTFILE, ">outfile");

close OUTFILE;
exit 0;
```

Try to fix the problem indicated by errno

If you attempt to continue after examining **`errno`**, you must reset it to zero in case you encounter a future error. On i5, **`#include unistd.h`** instead of the **`syscalls.h`** in K&R p. 171.

```
#include <errno.h>
#include <unistd.h>      /* for sleep */

    if (fp == NULL) {
        switch (errno) {

        case EMFILE:
            fclose(a file that I no longer need to keep open);
            break;

        case ENFILE:
            sleep(20);  /* Wait for other people to close their files */
            break;

        /* etc. */

        default:
            fprintf(stderr, "%s: unknown error %d\n", argv[0], errno);
            return EXIT_FAILURE;
        }

        errno = 0;
        try to fopen("outfile", "w") again;
```

Use a loop to keep trying. Terminate the loop when you succeed, or when you encounter an error that you can't attempt to fix, or when you've looped too many times.

```
#include <errno.h>
#include <unistd.h>

    while ((fp = fopen("outfile")) == NULL) {
        switch (errno) {

        case EMFILE:
            fclose(a file that I no longer need to keep open);
            break;

        case ENFILE:
```



```
35              sleep(20);   /* Wait for other people to close their files */
36              break;
37
38          /* etc. */
39
40          default:
41              fprintf(stderr, "%s: unknown error %d\n", argv[0], errno);
42              return EXIT_FAILURE;
43          }
44
45          errno = 0;
46      }
```

List a directory: Curry pp. 132−138; Bach pp. 73−74; K&R pp. 179−184

Every directory contains a file whose name is just a dot. For example, to see the dot file in your home directory,

```
1$ cd
2$ pwd

3$ ls -l                                  everything except those whose names begin with a dot
4$ ls -la                                 a for "all"
drwxr-xr-x  22 abc1234     users        4096 Jan  8 12:05 .
```

A dot file consists of sections called *entries* or *links*. Each entry contains the name and inode number of one of the files in the directory. The name is ASCII text but the inode number is in binary. Thus, like a **P6** format **.ppm** file, you can't **cat** a dot file onto the screen or **lpr** it.

In the early days of Unix, the inode number was 16 bits (see **ino_t** in KP p. 209) and the filename could be up to 14 characters (see **d_name** in KP p. 209 or Curry p. 34). Each entry was therefore the same length (16 bytes) and the dot file was simple enough so that you could read it with **od** (KP pp. 51, 59). But nowadays the inode number is 32 bits (see **ino_t** and **ulong_t** in **/usr/include/sys/types.h**) and the filename can be longer:

```
1 /* Excerpt from /usr/include/sys/fs/udf_inode.h */
2 #define  MAXNAMLEN    255
```

Each entry can now be a different length and the dot file is therefore so complicated that we have to call functions to read it: **opendir**, **readdir**, **closedir**. See **opendir**(3).

```
1 /* Excerpt from the file /usr/include/sys/types.h */
2 typedef unsigned long ulong_t
3 typedef ulong_t ino_t;              /* inode number */

4 /* Excerpt from the file /usr/include/sys/dirent.h, which is included by
5 /usr/include/dirent.h */
6
7 struct dirent {                        /* Each directory entry consists of: */
8         ino_t d_ino;           /* file number of entry */
9         char d_name[1];        /* any number of bytes */
10                               /* and more fields that I'm not showing you */
11 };
```

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/readdir.c**

```
1 /* Produce the same output as ls -fi */
2 #include <stdio.h>
```



```
 3 #include <stdlib.h>
 4 #include <errno.h>
 5 #include <sys/types.h>
 6 #include <dirent.h>
 7
 8 int main(int argc, char **argv)
 9 {
10     DIR *directory = opendir(".");
11     struct dirent *p;
12
13     if (directory == NULL) {
14         perror(argv[0]);
15         return EXIT_FAILURE;
16     }
17
18     while ((p = readdir(directory)) != NULL) {
19         printf("%6lu\t%s\n", p->d_ino, p->d_name);
20     }
21
22     /* Make sure we broke out of the while loop because of end-of-directory. */
23     if (errno != ESUCCESS) { /* Use 0 if you have no macro ESUCCESS. */
24         perror(argv[0]);
25         return EXIT_FAILURE;
26     }
27
28     if (closedir(directory) != 0) {
29         perror(argv[0]);
30         return EXIT_FAILURE;
31     }
32
33     return EXIT_SUCCESS;
34 }
```

You can split line 10 into the following lines 10 and 12

```
10     DIR *directory;
11     struct dirent *p;
12     directory = opendir(".");
```

—but why would you want to?

The operators in line 18 execute in the same order as those in the classic idiom in C

```
while ((c = getchar()) != EOF) {
```

When run in the directory **`/home1/m/mm64`**, the first lines output by the above program are:

```
   445      .
    16      ..
505379      .sh_history
114549      mail
 21560      bin
 65170      .profile
 19784      .jetadmin
 24254      math
 24263      public_html
 25085      46
```


▼ **Homework 1.3: fix the control structure. (not to be handed in)**

The **while** loop in the example in **readdir**(3) uselessly keeps checking to see if the value of **dirp** has become **NULL**. Fix it. It also calls **closedir** in two places. Fix it so that it calls **closedir** in only one place.

▲

The dot file is the directory itself.

Most people think that a *directory* is a section of the disk that holds a group of files, but this is an illusion. The files in a directory may actually be stored in widely separated places on the disk. The only thing these files have in common is that their names are listed in a dot file. In other words, a directory is just a dot file. In the above example, we could have changed

```
directory = opendir(".");
```

to

```
directory = opendir("/home1/m/mm64");
```

because **/home1/m/mm64** is just another name for this dot file.

The **ls** command merely displays the contents of a dot file (i.e., it outputs the **d_name** field of each entry in the dot file). The **rm** command merely removes an entry from a dot file, and then usually incinerates the file. The **mv** command (when used to rename a file) merely changes the **d_name** field of an entry in a dot file. The **mv** command (when used to move a file) merely removes an entry from one dot file and creates a new entry in another dot file.

Thus **mv** is really just a special-purpose editor. Why can't you **rm** or **mv** a file in a directory whose **w** bits are turned off?

```
1$ chmod 555 .          Change the current directory to r-xr-xr-x.
2$ ls -ld .             List the current directory itself, but not the files in it.
3$ rm file
```

Can you direct the output of a program into a dot file? Bach p. 74; KP p. 55

```
1$ date > outfile
2$ date >| outfile          override the Korn shell setnoclobber, ksh(1) pp. 15, 33

3$ cd
4$ pwd

5$ mkdir mydir
6$ ls -l | more
drwx------   2 abc1234     users        117 Jan  9 00:00 mydir

7$ cd mydir
8$ pwd
9$ ls -la
drwx------   2 abc1234     users        117 Jan  9 00:00 .

10$ date > .
.: Is a directory
```

The same error message results from


```
11$ cd
12$ pwd
13$ date > mydir                          mydir is just another name for this dot file.
mydir: Is a directory
```

Read a directory in Perl

The variable **$_** is the conventional variable for holding each successive line of input. Use **ne** to compare two strings, **!=** to compare two numbers:

http://i5.nyu.edu/~mm64/x52.9544/src/readdir

```
#!/bin/perl
#Output the names of all the things in the specified directory,
#one per line.

opendir(DIRHANDLE, '/home/m/mm64') || die "$0: $!";

while (($_ = readdir(DIRHANDLE)) ne '') {
    print "$_\n";
}

closedir(DIRHANDLE) || die "$0: $!";
exit 0;
```

The operators in the **while** expression execute in the same order as those in the classic idiom in C

```
while ((c = getchar()) != EOF) {
```

You can change it to

```
while ($_ = readdir(DIRHANDLE)) {
```

File types: Curry pp. 105−107, 110−111; Bach pp. 61, 88; KP pp. 54, 66, 214−215; K&R pp. 181−182

The first character on each line output by **ls -l** shows the type of the file. The seven possibilities are listed in **ls**(1), **stat**(2), and the seven **S_IF #define**'s in **/usr/include/sys/stat.h** (or **mode.h** in other versions of Unix).

In other versions, typical terminal names might be **/dev/ttypa** or **/dev/pts/10**.

/devices/pseudo/pts@0:0	*character device, e.g., a terminal*
/dev/vx/dsk/rootvol	*block device, e.g., a disk drive*
/bin/perl	*symbolic link, created by* **ln -s**
/var/spool/lp/fifos/FIFO	*named pipe, i.e., FIFO, created by* **mknod p**
/tmp/mysql.sock	*socket, created by* **socket** *system call*
/home1/a	*directory, i.e., a dot file, created by* **mkdir**
/etc/passwd	*none of the above, i.e., a plain, regular file*

```
1$ ls -l
crw--w----   1 root     tty       24,  0 Dec 14  2000 pts@0:0
brw-------   1 root     root     200,  0 Dec 14  2000 rootvol
lrwxrwxrwx   1 root     other         19 Dec 15  2000 perl -> /usr/local/bin/perl
prw-rw-rw-   1 lp       lp             0 Dec 14  2000 FIFO
drwxr-xr-x 951 root     other      20480 Oct 22 10:29 a
-r--r--r--   1 root     sys       736431 Jan  5 15:19 passwd
```

```
2$ find / -type p -print 2> /dev/null | more
```

Inodes: Curry pp. 109−110, 112−121 Bach pp. 61−63; KP pp. 57−63, 214−219; K&R pp. 179−182

All the facts about a file (other than its name and contents) are stored in a structure called the file's *inode*. Each file has one inode, and each inode has an identifying number. To see the inode number as well as the name of each file in a directory, run the **opendir** C program shown above or say **ls -i** or **ls -li** (minus lowercase LI).

To read the fields of an inode, declare a **struct stat** and pass the structure's address to the system call **stat** to fill it up. See **stat**(2). You don't have to give **stat** a full pathname: like all Unix programs, it assumes that the file is in the current directory if you specify no path.

The **ls -l** command merely reads and displays the contents of the inode of each file listed in a dot file.

```
 1 /* Excerpts from the file /usr/include/sys/types.h */
 2
 3 typedef ulong_t dev_t;        /* contains the major and minor device numbers */
 4 typedef ulong_t major_t;
 5 typedef ulong_t minor_t;
 6 typedef ulong_t mode_t;       /* permission bits: rwxrwxrwx */
 7 typedef ulong_t nlink_t;      /* number of hard links coming out of a file */
 8 typedef long uid_t;           /* user ID number (field 3 of /etc/passwd) */
 9 typedef uid_t gid_t;          /* group ID number (field 3 of /etc/group) */
10 typedef long off_t;           /* file's size in bytes */
11 typedef long time_t;          /* number of seconds since Midnight 1/1/70 */

12 /* Excerpts from the file /usr/include/sys/stat.h */
13
14 struct stat {                 /* excerpts */
15     dev_t   st_dev;           /* major and minor device numbers */
16     ino_t   st_ino;           /* inode number */
17     mode_t  st_mode;          /* the permission bits: rwxrwxrwx */
18     nlink_t st_nlink;         /* how many hard links come out of this file */
19     uid_t   st_uid;           /* UID number of file's owner */
20     gid_t   st_gid;           /* GID number of file's owner's group */
21     off_t   st_size;          /* size of file in bytes */
22     time_t  st_atime;         /* last time accessed (fed as input) ls -lu */
23     time_t  st_mtime;         /* last time modified (edited) ls -lt */
24     time_t  st_ctime;         /* last time changed (edited or chmod'ed) ls -lc */
25 };

26 /* Excerpts from the file /usr/include/sys/stat.h */
27
28 /* Permission bits.  See stat(2), chmod(2). */
29
30 #define S_IAMB 0x1FF          /* access mode bits: the nine permission bits */
31 #define S_IRUSR 00400         /* read permission: user (i.e., owner) */
32 #define S_IWUSR 00200         /* write permission: user */
33 #define S_IXUSR 00100         /* execute permission: user */
34
35 #define S_IRGRP 00040         /* read permission: group */
36 #define S_IWGRP 00020         /* write permission: group */
37 #define S_IXGRP 00010         /* execute permission: group */
38
```

```
39 #define S_IROTH 00004        /* read permission: other (i.e., everybody else) */
40 #define S_IWOTH 00002        /* write permission: other */
41 #define S_IXOTH 00001        /* execute permission: other */
42
43 /* type of file */
44 #define S_IFMT   0xF000
45
46 #define S_IFBLK  0x6000      /* b  block device, e.g., a disk drive in /dev */
47 #define S_IFCHR  0x2000      /* c  character device, e.g., a terminal in /dev */
48 #define S_IFLNK  0xA000      /* l  symbolic link, created by ln -s */
49 #define S_IFIFO  0x1000      /* p  FIFO, i.e., named pipe, created by mknod p */
50 #define S_IFSOCK 0xC000      /* s  socket, created by socket system call */
51 #define S_IFDIR  0x4000      /* d  directory, i.e., a dot file, created by mkdir */
52 #define S_IFREG  0x8000      /* -  regular file, i.e., none of the above */
```

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/stat.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4 #include <sys/types.h>
 5 #include <sys/stat.h>    /* declarations for the st_ fields */
 6 #include <sys/mkdev.h>   /* declarations for major and minor functions */
 7
 8 int main(int argc, char **argv)
 9 {
10     struct stat status;
11     mode_t type;
12     char age[256];
13
14     if (stat("/etc/passwd", &status) != 0) {
15         perror(argv[0]);
16         return EXIT_FAILURE;
17     }
18
19     printf("%lu\tinode number\n", status.st_ino);
20
21     printf("%08lX\tdevice number\n", status.st_dev);
22     printf("%lu\tmajor device number\n", major(status.st_dev));
23     printf("%lu\tminor device number\n", minor(status.st_dev));
24
25     printf("%lu\tmode\n", status.st_mode);
26     type = status.st_mode & S_IFMT;
27     printf("%lu\tfile type\n", type);
28
29     if (type == S_IFBLK) {
30         printf("This is a block device.\n");
31     } else if (type == S_IFCHR) {
32         printf("This is a character device.\n");
33     } else if (type == S_IFLNK) {
34         printf("This is a symbolic link.\n");
35     } else if (type == S_IFIFO) {
36         printf("This is a named pipe.\n");
37     } else if (type == S_IFSOCK) {
38         printf("This is a socket.\n");
```



```
39      } else if (type == S_IFDIR) {
40          printf("This is a directory.\n");
41      } else if (type == S_IFREG) {
42          printf("This is a regular file.\n");
43      } else {
44          printf("unknown file type %lu.\n", type);
45      }
46
47      printf("%03lo\tnine permission bits (in octal)\n", status.st_mode & S_IAMB);
48
49      printf("%lu\tnumber of links\n", status.st_nlink);
50      printf("%ld\tfile's UID number\n", status.st_uid);
51      printf("%ld\tfile's GID number\n", status.st_gid);
52      printf("%ld\tsize in bytes\n", status.st_size);
53
54      printf("%ld\tseconds since Midnight 1/1/70\n", status.st_mtime);
55      strftime(age, sizeof age, "%b %d %H:%M %Y", localtime(&status.st_mtime));
56      printf("%s\n", age);
57
58      return EXIT_SUCCESS;
59  }
```

Here's the output:

```
318949      inode number
03200000    device number
200         major device number
0           minor device number
33060       mode
32768       file type
This is a regular file.
444         nine permission bits (in octal)
1           number of links
0           file's UID number
3           file's GID number
736431      size in bytes
1073333952  seconds since Midnight 1/1/70
Jan 05 15:19 2004
```

```
1$ cd /etc
2$ ls -li passwd
    318949 -r--r--r--   1 root     sys       736431 Jan  5 15:19 passwd

3$ awk -F: '$3 == 0 {print $1}' /etc/passwd
root

4$ awk -F: '$3 == 3 {print $1}' /etc/group
sys

5$ bc                                   binary calculator
scale=5
1073333952 / 60 / 60 / 24 / 365.25
34.01190
control-d
```



```
6$ df /etc/passwd                                    disk free
/                  (/dev/vx/dsk/rootvol):  562598 blocks   426744 files

7$ ls -l /dev/vx/dsk/rootvol
brw-------   1 root     root     200,  0 Dec 14  2000 /dev/vx/dsk/rootvol
```

Examples of bitwise &: K&R pp. 48−49

The following **if** statement is true for **/etc/motd**. Use parentheses to execute the **&** before the **==** as in K&R, pp. 181−182.

```
41      } else if (type == S_IFREG) {

            type     1000 000110100100
        & S_IFMT     1111 000000000000
         S_IFREG 1000     000000000000

47      printf("%03lo\tnine permission bits (in octal)\n", status.st_mode & 0777);

          status.st_mode     1000000 110100100
        &         S_IAMB     0000000 111111111
               rw-r--r-- 0000000     110100100
```

stat in Perl

The **->** makes the **$st** look like a structure and makes the **ino** look like a structure field. But the **$st** is really a **File::stat** object and the **ino** is really a method of that object with no arguments.

http://i5.nyu.edu/~mm64/x52.9544/src/stat

```
#!/bin/perl

use File::stat;

$st = stat('/etc/passwd');
defined $st || die "$0: $!";

print $st->ino, "\tinode number\n",
    $st->dev, "\tdevice number\n",
    $st->mode, "\tmode\n";

exit 0;
```

```
318949      inode number
52428800    device number
33060       mode
```

▼ Homework 1.4: write a program to list the files in the current directory

Write a C, C++, or Perl program named **myls** to produce the same output as

```
ls -lai | tail +2
```

Hand in the output whe run in the following directory:

```
1$ cd $S44/dir
2$ pwd
/home1/m/mm64/public_html/x52.9544/src/dir
```

The directory **/home1/m/mm64/public_html/x52.9544/src** is on the web at **http://i5.nyu.edu/~mm64/x52.9544/src/**.

```
3$ myls
 24381 drwxr-xr-x   5 mm64     users       4096 Mar 11  2001 .
 24374 drwxr-xr-x   3 mm64     users       4096 Jan  8 00:51 ..
 24382 drwxr-xr-x   2 mm64     users         96 Jan 31  2001 dir1
 24383 drwsr-xr-t   2 mm64     users         96 Jan 31  2001 dir2
 24384 drwSrw-rwT   2 mm64     users         96 Jan 31  2001 dir3
 24385 -r-xr-xr-x   2 mm64     users          6 Jan 27  1998 file1
 24385 -r-xr-xr-x   2 mm64     users          6 Jan 27  1998 hard_link
 24386 -r-sr-sr-x   1 mm64     users          0 Jan 27  1998 file3
 24387 -r-Sr-lr--   1 mm64     users          0 Jan 27  1998 file4
 24388 lrwxrwxrwx   1 mm64     users         15 Jan 31  2001 symbolic_link -> /usr/dic
 24389 p---------   1 mm64     users          0 Jan 27  1998 pipe
 24390 ----------   1 mm64     users          0 Jan 27  1998 file2
```

I created the named pipe with the command

```
4$ /usr/sbin/mknod pipe p
```

Do not write the name of the **$S44/dir** directory in your program: the directory that your program lists must be its *current* directory. You get no credit if your program takes any command line arguments. You get no credit if your program calls **chdir** or contains the name of the **$S44/dir** directory.

Loop through all the names in the dot file of the current directory, and pass each name to **stat**. Do not sort the names in alphabetical order: simply output them in the order in which **readdir** returns them to you. Right-justify the inode number in a field at least 6 characters wide, (i.e., **printf "%6lu"**) and right-justify the file size in a field at least 10 characters wide. When the loop is over, **closedir** and **return** from **main** with exit status **EXIT_SUCCESS**.

Hand the C, C++, or Perl program itself and the program's output when run in the directory **$S44/dir**. You get no credit without the output.

Print the file type

The first character of the second field on each output line must be one of the seven filetype characters **-bcdlps**. Do not write seven **if** statements like the seven shown above. You get credit only if you loop with a pointer through an array of structures:

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/st_mode.c**

```
 1 /* Output the filetype of /etc/passwd. */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <time.h>
 5 #include <sys/types.h>
 6 #include <sys/stat.h>
 7
 8 typedef struct {
 9     char c;          /* the first character on each line output by ls -l */
10     mode_t mask;     /* one of the S_IF macros */
11 } filetype_t;
12
13 int main(int argc, char **argv)
```



```
14 {
15     filetype_t a[] = {
16         {'b',   S_IFBLK},   /* block device */
17         {'c',   S_IFCHR},   /* character i/o device */
18         {'l',   S_IFLNK},   /* symbolic link */
19         {'p',   S_IFIFO},   /* named pipe */
20         {'s',   S_IFSOCK},  /* socket */
21         {'d',   S_IFDIR},   /* directory */
22         {'-',   S_IFREG},   /* regular file */
23
24         {'\0',  0}          /* end of data */
25     };
26
27     struct stat status;
28     filetype_t *p;
29
30     if (stat("/etc/passwd", &status) != 0) {
31         perror(argv[0]);
32         return EXIT_FAILURE;
33     }
34
35     for (p = a; p->mask != 0; ++p) {
36         if ((status.st_mode & S_IFMT) == p->mask) {
37             printf("%c\n", p->c);
38             RETURN EXIT_SUCCESS;
39         }
40     }
41
42     fprintf(stderr, "%s: unknown file type, mode == %06lo\n",
43         argv[0], status.st_mode);
44     return EXIT_FAILURE;
45 }
```

```
    -                                                    It outputs a dash.
```

You get credit only if you print the nine permission bits by looping through an array of nine structures with a pointer to a structure. Do not write nine **if** statements. The array of nine structures must contain 'single-quoted' characters, not "double-quoted" strings.

Print the file type in Perl

Just use the seven operators **-b**, **-c**, **-l**, **-p**, **-S** (uppercase), **-d**, **-f** (normal file).

```
#!/bin/perl

if (-b '/etc/passwd') {
    print "b\n";
} elsif (-f '/etc/motd') {
    print "-\n";
}

exit 0;
```

```
    -
```


Print the loginname of the file's owner

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/getpwuid.c**

```
 1 /* Output the loginname of the usr with UID number 50766. */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <pwd.h>
 5
 6 int main(int argc, char **argv)
 7 {
 8     uid_t uid = 50766;
 9     struct passwd *p = getpwuid(uid);
10
11     if (p == NULL) {
12         fprintf(stderr, "%s: UID %u does not exist\n", argv[0], uid);
13         return EXIT_FAILURE;
14     }
15
16     printf("%u %s\n", uid, p->pw_name);
17     return EXIT_SUCCESS;
18 }
```

```
    50766 mm64
```

Print the loginname of the file's owner in Perl

http://i5.nyu.edu/~mm64/x52.9544/src/getpwuid

```
#!/bin/perl

@password = getpwuid(50766);
defined @password || die "$0: $!";
print "$password[0]\n";

exit 0;
```

Print the name of the group

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/getgrgid.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <grp.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     gid_t gid = 15;
 8     struct group *p = getgrgid(gid);
 9
10     if (p == NULL) {
11         fprintf(stderr, "%s: GID %u does not exist\n", argv[0], gid);
12         return EXIT_FAILURE;
13     }
14
15     printf("%u %s\n", gid, p->gr_name);
16     return EXIT_SUCCESS;
```



```
17 }
```

```
    15 users
```

Print the name of the file's owner's group in Perl

http://i5.nyu.edu/~mm64/x52.9544/src/getgrgid

```
#!/bin/perl

@group = getgrgid(15);
defined @group || die "$0: $!";
print "$group[0]\n";

exit 0;
```

```
    users
```

✎ **Display old files differently**

Extra credit. **ls -l** displays the year instead of the hour and minute when a file is more than six months old:

```
    1$ cd /etc
    2$ ls -lt | more
    -r--r--r--   1 root     staff      26970 Jan  8 14:18 group
    -rw-r--r--   1 root     sys         1802 Jan  5  2000 dacf.conf
```

To find the current time, measured in seconds since Midnight, January 1, 1970,

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/strftime.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     time_t current;
 8     char age[256];
 9
10     time(&current);
11     printf("current == %d\n", current);
12     strftime(age, sizeof age, "%b %d %H:%M %Y", localtime(&current));
13     printf("%s\n", age);
14
15     return EXIT_SUCCESS;
16 }
```

```
    current == 1073624445
    Jan 09 00:00 2004
```

To see how old **/etc/passwd** is,

```
     1073624445
    -1073333952
         290493
```

There are 15,768,000 seconds in six months, but please write **60 * 60 * 24 * 365 / 2** instead of

15768000:

```
/* Display year instead of hour & minute for files older than this
number of seconds. */
const time_t threshold = 60 * 60 * 24 * 365 / 2;
```

The word **strftime** must appear only once in your program.

The elements of the array returned by the Perl **localtime** function are just like the fields of the C **struct tm**. See K&R, p. 255.

http://i5.nyu.edu/~mm64/x52.9544/src/localtime

```
#!/bin/perl

print time(), "\n";

@F = localtime(time());  #or just @F = localtime();

printf "month == %d, day == %d, %02d:%02d\n",
    $F[4], $F[3], $F[2], $F[1];

exit 0;
```

```
1073624446
month == 0, day == 9, 00:00
```

✎ **Display the three extra permission bits: Bach pp. 225−229; Curry pp. 124−125; KP pp. 54−55**

```
1$ cd /bin
2$ ls -l at mailx passwd
-rwsr-xr-x   1 root     sys        37784 Aug 27  2002 at
-r-x--s--x   1 root     mail      126880 Jan  9  2002 mailx
-r-xr-xr-x   1 root     other        555 Feb 20  2002 passwd
```

Extra credit. In addition to the nine classic permission bits, display the values of the **S_ISUID**, **S_ISGID**, and **S_ISVTX** bits listed in **/usr/include/sys/stat.h**. See **ls**(1).

(1) When **S_ISUID** and **S_IXUSR** are both on, print a lowercase **s** in place of the user's execute permission. When **S_ISUID** is on and and **S_IXUSR** is off, print an uppercase **S** in place of the user's execute permission.

(2) When **S_ISGID** and **S_IXGRP** are both on, print a lowercase **s** in place of the group's execute permission. When **S_ISGID** is on and and **S_IXGRP** is off, print an uppercase **S** in place of the group's execute permission.

(3) When **S_ISVTX** and **S_IXOTH** are both on, print a lowercase **t** in place of the other people's execute permission. When **S_ISVTX** is on and and **S_IXOTH** is off, print an uppercase **T** in place of the other people's execute permission. For example,

```
3$ cd
4$ pwd
5$ date > junk

6$ chmod 4555 junk
7$ ls -l
-r-sr-xr-x   1 abc1234     users         29 Jan  9 00:00 junk
```



```
8$ chmod 4455 junk
9$ ls -l
-r-Sr-xr-x   1 abc1234     users         29 Jan  9 00:00 junk

10$ mkdir junkdir            we can use the S_ISVTX bit only for directories, not files

11$ chmod 1555 junkdir
12$ ls -ld junkdir
dr-xr-xr-t   2 abc1234     users        117 Jan  9 00:00 junkdir

13$ chmod 1554 junkdir
14$ ls -ld junkdir
dr-xr-xr-T   2 abc1234     users        117 Jan  9 00:00 junkdir
```

```
#!/bin/perl

if (-u '/etc/passwd') {
    print "s\n";
} else {
    print "-\n";
}

exit 0;
```

```
-
```

✎ **Print the major and minor device numbers**

Extra credit. If the file is of type **S_IFBLK** or **S_IFCHR** (i.e., **b** or **c**), print the major and minor device numbers (separated by a comma) instead of the size. Take this information from the **st_rdev** field of the **struct stat**. Use the **major** and **minor** macros in **/usr/include/sys/types.h**.

```
1$ ls -l /etc/passwd /dev/vx/dsk/rootvol /devices/pseudo/pts@0:0
brw-------   1 root     root     200,  0 Dec 14  2000 /dev/vx/dsk/rootvol
crw--w----   1 root     tty       24,  0 Dec 14  2000 /devices/pseudo/pts@0:0
-r--r--r--   1 root     sys       736431 Jan  5 15:19 /etc/passwd
```

http://i5.nyu.edu/~mm64/x52.9544/src/major

```
#!/bin/perl
#The top 14 bits are the major device number.
#The bottom 18 bits are the minor device number.
use File::stat;

$st = stat('/dev/vx/dsk/rootvol');
defined $st || die "$0: $!";

$d = $st->rdev;
print "major == ", $d >> 18 & 0x3FFF,
    ", minor == ", $d & 0x3FFFF, "\n";

exit 0;
```

```
major == 200, minor == 0
```

✎ **Print the name of the referenced file**

Extra credit. If the file you're investigating is a symbolic link (e.g., **newname** below), **stat** retrieves information about the referenced file while **lstat** retrieves information about the link itself (i.e., the little one-line file which contains the name of the referenced file). Do not call **stat** anywhere in your program. Always call **lstat**, and if the file is of type **S_IFLNK**, print an arrow **->** and the name of the referenced file:

```
1$ cd
2$ pwd

3$ date > oldname
4$ ls -l

5$ ln -s /home1/a/abc1234/oldname newname          The filename is 23 characters.
6$ ls -l
lrwxrwxrwx   1 abc1234     users         28 Jan  9 00:00 newname -> /home1/a/abc1234/oldna
-rw-------   1 abc1234     users         29 Jan  9 00:00 oldname
```

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/lstat.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4 #include <sys/types.h>
 5 #include <sys/stat.h>
 6 #include <unistd.h>
 7
 8 int main(int argc, char **argv)
 9 {
10     struct stat status;
11     char *p;
12     int n;          /* number of characters read in */
13
14     if ( /* the vampire */ lstat("newname", &status) != 0) {
15         perror(argv[0]);
16         return EXIT_FAILURE;
17     }
18
19     if ((status.st_mode & S_IFMT) == S_IFLNK) {
20         printf("Symbolic link containing %d characters:\n", status.st_size);
21         p = malloc(status.st_size + 1);
22         if (p == NULL) {
23             perror(argv[0]);
24             return EXIT_FAILURE;
25         }
26
27         n = readlink("newname", p, status.st_size);
28         if (n < 0) {
29             perror(argv[0]);
30             return EXIT_FAILURE;
31         }
32
33         p[n] = '\0';
34         printf("-> %s\n", p);
35         free(p);
```



```
36      }
37
38      return EXIT_SUCCESS;
39  }
```

```
    Symbolic link containing 23 characters:
    -> /home1/a/abc1234/oldname
```

You can combine lines 21−31 to

```
21          if ((p = malloc(status.st_size + 1)) == NULL ||
22              (n = readlink("newname", p, status.st_size)) < 0) {
23              perror(argv[0]);
24              return EXIT_FAILURE;
25          }
```

✎ **Print the total number of blocks**

Extra credit. I showed you only my favorite fields of the **`struct stat`**. One of the ones I didn't show was **`st_blocks`**. As you loop through every entry in the directory, add up the **`st_blocks`** field of each entry. Store the sum in a variable whose data type is the same as that of the **`st_blocks`** field. When you're done, print the total divided by 2. Prerequisite: the **`lstat`** extra credit.

```
1$ cd $S44/dir
2$ ls -la | head -1
total 32
```

▲

□