Lecture #8: Standard I/O Library

Reading: *none*

Handouts: Print (Pseuco)random Saying

---

1. Greetings and felicitations
2. Structure: a portable interface to the underlying system calls
   - how it represents files (streams with an offset); FILE *
   - stdin, stdout, stderr
3. Opening and Closing
   - fopen("zyx", "r"); mode is r (read), w (write), a (append), r+ (read and write, file offset is 0), w+ (read and write, truncate file if it exists), a+ (w+ but append and not truncate); note some systems have "b" appended as binary; why?
   - freopen("file", "r", stdin); now reads from stdin really read from file
   - fclose(fp); close file pointer
4. Reading and Writing Lines
   - formatted output: printf, fprintf, sprintf (cover d, x, o, f, e, g, c, s)
   - formatted input: scanf, sscanf, fscanf (cover d, x, o, f, [], c, s)
   - fgets, gets, fputs, puts (remember: f keeps newline or adds it)
5. Reading and Writing Characters
   - getc, getchar, fgetc
   - putc, putchar, fputc
   - ungetc
6. Reading and Writing Blocks
   - fread(ptr, size, number, fp), fwrite(ptr, size, number, fp)
7. Random I/O
   - fseek(fp, offset, whence); rewind(fp) same as fseek(fp, 0L, SEEK_SET); relative positioning OK; need errno to find error
   - ftell(fp)
8. Go through randmsg

Print (Pseudo)random Saying

```
/*
 * PRS -- generate a pseudorandom saying
 *
 * use: prs file          --- print saying chosen at random from file
 *      prs -b file       --- put file into the right format
 *                            (file is a sequence of sayings
 *                            separated by a single line ENDSAYING;
 *                            only first MAXSAYING sayings are used
 *    prs -d file  --- print out debugging information about
 *                            file
 * author: Matt Bishop, bishop@cs.ucdavis.edu
 */
#include <stdio.h>
#include <errno.h>
#include <sys/time.h>
#include <unistd.h>

/*
 * file format
 * <the sayings, separated with a line containing only "---">
 * <the index table, each long being the offset from the
 *  beginning of the file; all entries unsigned long>
 * <the number of entries in the index table, unsigned long>
 * <the magic number MAGIC, unsigned long>
 *
 * beginning of file --->        text saying #1
 *                               ---
 *                               text saying #2
 *                               <0 as an unsigned long>
 *                               <20 as an unsigned long>
 *                               <2 as an unsigned long>
 *                               <35 as an unsigned long>
 * end of file --------->        <MAGIC as an unsigned long>
 */
#define MAGIC 0x01030289  /* the magic number */
unsigned long magic = MAGIC;     /* variable for magic */
unsigned long npos;       /* size of index table */
unsigned long tab;        /* position of index table */
char *progname = "prs";   /* program name */

/*
 * macros and such
 */
#define ENDSAYING  "---\n"       /* end of saying; MUST end in \n */
#define MAXSAYING  1024          /* max number of sayings per file */
#define MAXLINE    1024          /* max line length */
#define DEBUG_COLS 5             /* this many offsets/col for debug */
#define DEFFILE    "/home/ecs40r/lib/sayings" /* default file */

/*
 * types and such
 */
enum eact {
      MAKE,        /* create a randsay file */
```

```
      DEBUG,       /* print out a randsay file in full */
      SHOW,        /* show a saying */
};
extern int optind; /* next arg index to be processed */
                        /* usage message */
#define USAGE      fprintf(stderr, "prs [ -b | -d ] [ file ]\n")

/*
 * unprototyped system functions (sigh)
 */
int gettimeofday(struct timeval *, struct timezone *);
int getopt(int, char **, char *);
void srandom(int);
long random(void);

/*
 * generate random saying file (see above for format)
 * arguments:      FILE *fp      points to saying file
 * actions: generates a sayings file for this program
 * errors: * if it can't write out the binary data
 *          * too many sayings -- only first MAXSAYING used
 */
void makefile(FILE *fp)
{
      unsigned long pos[MAXSAYING];      /* array of indices */
      char saying[MAXLINE];              /* saying */
      register int i;                    /* counter in a for loop */

      /*
       * Process the file, recording  offset into the
       * position array
       */
      for(npos = 0; npos < MAXSAYING && !feof(fp); npos++){
            /* get saying position */
            pos[npos] = ftell(fp);
            /* skip over the saying */
            i = 0;
            while (fgets(saying, MAXLINE, fp) != NULL &&
                        strcmp(ENDSAYING, saying) != 0)
                  i++;
            /* ignore empty sayings */
            if (i == 0)
                  npos--;
      }
      /*
       * array will be appended to the file
       */
      errno = 0;
      if (fseek(fp, 0L, SEEK_END) != -1 && errno != 0){
            fprintf(stderr, "Can't find end of file.\n");
            return;
      }
      tab = ftell(fp);
      /*
       * now write out the index table, size, index
       * table location, and magic number
       */
      if (fwrite(pos, sizeof(unsigned long), npos, fp) != npos ||
```

```
                fwrite(&npos, sizeof(npos), 1, fp) != 1 ||
                fwrite(&tab, sizeof(tab), 1, fp) != 1 ||
                fwrite(&magic, sizeof(magic), 1, fp) != 1){
            fprintf(stderr, "Could not write index.\n");
            return;
        }
}


/*
 * print debugging information
 * arguments:      FILE *fp      points to saying file
 *                 char *fname   file name
 * actions: prints the sayings plus positional information
 * errors: * barfs if magic number wrong or missing
 *          * barfs if can't find the table, index, or
 *            the string
 */
void debugfile(FILE *fp, char *fname)
{
      unsigned long pos;  /* current position in file */
      char saying[MAXLINE];
      int i, k;

      /*
       * print the magic number, index table,
       * and index table size
       */
      errno = 0;
      if ((fseek(fp, -(sizeof(magic)+sizeof(npos)+sizeof(tab)),
                  SEEK_END) == -1 && errno != 0) ||
                  fread(&npos, sizeof(npos), 1, fp) != 1 ||
                  fread(&tab, sizeof(tab), 1, fp) != 1 ||
                  fread(&magic, sizeof(magic), 1, fp) != 1){
            perror(fname);
            return;
      }
      printf("magic humber 0x%08lx\ntable at 0x%08lx, %ld entr%s\n",
            magic, tab, npos, npos == 1 ? "y" : "ies");
      /*
       * now print out the index table entries, 5 per line
       */
      errno = 0;
      if (fseek(fp, tab, SEEK_SET) == -1 && errno != 0){
            perror(fname);
            return;
      }
      for(i = k = 0; i < npos; i++){
            /* read the index entry; error if you fail */
            if (fread(&pos, sizeof(unsigned long), 1, fp) != 1){
                  printf(" ********");
                  perror(fname);
                  continue;
            }
            /* print it out */
            printf(" 0x%08lx", pos);
            /* newline if appropriate */
            if (k == DEBUG_COLS){
```

```
                putchar('\n');
                k = 0;
            }
            else
                k++;
        }
        /*
         * print the sayings, led by offsets and without terminators
         */
        if (i != 0)
            putchar('\n');
        (void) rewind(fp);
        while(!feof(fp) && (pos = ftell(fp)) < tab){
            /* print offset */
            printf("offset 0x%08lx:\n", pos);
            /* print saying */
            while(fgets(saying, MAXLINE, fp) != NULL &&
                        strcmp(ENDSAYING, saying) != 0)
                printf("   %s", saying);
        }
}

/*
 * print message
 * arguments:      FILE *fp      points to saying file
 *                 int n         number to compute  index from
 * actions: prints the saying
 * errors: * barfs if magic number wrong or missing
 *          * barfs if can't find the table, index, or
 *            the string
 */
void sayprint(FILE *fp, int n)
{
        char saying[MAXLINE];             /* buffer for saying */

        /*
         * get magic number
         */
        errno = 0;
        if ((fseek(fp, -sizeof(magic), SEEK_END) == -1 && errno != 0) ||
                        fread(&magic, sizeof(magic), 1, fp) != 1){
            fprintf(stderr, "Can't read magic number.\n");
            return;
        }
        /*
         * is the magic number right?
         */
        if (magic != MAGIC){
            fprintf(stderr, "Magic number wrong.\n");
            return;
        }
        /*
         * get the size and address (offset) of the table
         * and position pointer at this place
         */
        errno = 0;
        if ((fseek(fp, -(sizeof(magic)+sizeof(npos)+sizeof(tab)),
                SEEK_END) == -1 && errno != 0) ||
```

```
                fread(&npos, sizeof(npos), 1, fp) != 1 ||
                fread(&tab, sizeof(tab), 1, fp) != 1){
            fprintf(stderr, "Can't find index or index size\n");
            return;
        }
        /*
         * get the address of the saying to print
         */
        errno = 0;
        if ((fseek(fp, tab+ (n % npos) * sizeof(unsigned long),
                SEEK_SET) == -1 && errno != 0) ||
                fread(&tab, sizeof(tab), 1, fp) != 1){
            fprintf(stderr, "Can't find address of saying\n");
            return;
        }
        /*
         * get the saying itself and print it (phew!)
         */
        errno = 0;
        if (fseek(fp, tab, SEEK_SET) == -1 && errno != 0){
            fprintf(stderr, "Can't get to saying\n");
            return;
        }
        while(fgets(saying, MAXLINE, fp) != NULL &&
                        strcmp(saying, "---\n") != 0)
            printf("%s", saying);
}

/*
 * seeds the random number generator
 * arguments:      none
 * actions: uses microseconds, PID, and parent PID to
 *           seed the system's pseudorandom number generator
 * errors: * can't get time of day (should never happen)
 */
void seedrng(void)
{
        struct timeval tod; /* use time of day (usec) */
        struct timezone tz; /* time zone (unused) */

        /* begin with microseconds */
        if (gettimeofday(&tod, &tz) < 0)
            perror("gettimeofday");
        /* mix with PID and parent PID */
        tod.tv_usec ^= ((getpid()<<16)|getppid());
        srandom((int) tod.tv_usec);
}

/*
 * the main routine
 */
int main(int argc, char *argv[])
{
        register FILE *fp;      /* ptr to input/saying file */
        register int c;                 /* the option */
        enum eact action = SHOW;  /* what to do */

        /*
```

```
 * set the program name
 */
progname = argv[0];

/*
 * do the options
 */
while((c = getopt(argc, argv, "bd")) != EOF){
      switch(c){
      case 'b':             /* build */
             action = MAKE;
             break;
      case 'd':             /* debug */
             action = DEBUG;
             break;
      case '?':             /* huh? */
             USAGE;
             return(0);
      }
}

/*
 * everything from here on requires
 * a file, so open it
 */
if (optind == argc)
      /* no file named -- use default */
      argv[optind] = DEFFILE;
if ((fp = fopen(argv[optind], "r+")) == NULL){
      perror(argv[optind]);
      return(1);
}

/*
 * now do it!
 */
switch(action){
case MAKE:                  /* make the file */
      makefile(fp);
      break;
case DEBUG:                 /* print debugging version */
      debugfile(fp, argv[optind]);
      break;
case SHOW:                  /* print message */
      /* generate random number and print it */
      seedrng();
      sayprint(fp, (int) random());
      break;
default:                    /* ??? */
      fprintf(stderr, "Internal error: action ILLEGAL (%d)\n",
                              action);
      break;
}

/*
 * bye!
 */
return(0);
```

```
}
```