CHAPTER 18

EXTERNAL FILES

18.1 ACCESSING EXTERNAL FILES
- Character File I/O
- Formatted File I/O
- Line-Oriented File I/O

18.2 THE STANDARD FILES

18.3 RANDOM FILE ACCESS

18.4 BLOCK INPUT AND OUTPUT

18.5 FILE UPDATING

18.6 CASE STUDY—AN ELECTRONIC ADDRESS BOOK

filecopy.c

(Figure 18.1 – Page 447)

```
/*
 * File copying utility.  Usage is: filecopy source dest.
 */
#include <stdio.h>
#include <string.h>
#include <stddef.h>

int main(int argc, char *argv[])
{
  long filecpy(const char *dest, const char *source);
  long copycnt;                    /* number of characters copied */
  int  status = 1;                 /* program return value */

  if (argc != 3)
    printf("Usage: %s source dest\n", argv[0]);
  else if ((copycnt = filecpy(argv[2], argv[1])) == -1L)
    printf("Copy failed\n");
  else
  {
    printf("Copied %li characters\n", copycnt);
    status = 0;
  }
  return status;
}
```

filecpy1.c

(Figure 18.2 – Page 448)

```
/*
 * Actually copy one file into another.
 */
#include <stdio.h>
#include <stddef.h>

long filecpy(const char *dest, const char *source)
{
  FILE *sfp;                    /* source file pointer */
  FILE *dfp;                    /* destination file pointer */
  int  c;                       /* next input character */
  long cnt = -1L;               /* count of characters copied */

  if ((sfp = fopen(source,"rb")) == NULL)
    printf("Can't open %s for reading\n", source);
  else
  {
    if ((dfp = fopen(dest,"wb")) == NULL)
      printf("Can't open %s for writing\n", dest);
    else
    {
      for (cnt = 0L; (c = getc(sfp)) != EOF; cnt++)
        putc(c,dfp);
      fclose(dfp);
    }
    fclose(sfp);
  }
  return cnt;
}
```

fints.c

(Figure 18.3 – Page 450)

```
/*
 * Read up to "max" values into "a" from file "name".
 */
#include <stdio.h>
#include "getints.h"

#define TESTFILE  "table"
#define MAX       20

int main()
{
  int table[MAX], i;
  int status = 1;

  for (i = 0; i < MAX; i++)        /* some sample values to write */
    table[i] = i;
  if (fputints(TESTFILE, table, MAX) != MAX)
    printf("Couldn't write array to file\n");
  else if (fgetints(TESTFILE, table, MAX) != MAX)
    printf("Couldn't read array from file\n");
  else
  {
    printf("Wrote and read %i ints using %s\n", MAX, TESTFILE);
    status = 0;
  }
  return status;
}
```

getints.h

(Figure 18.4 – Page 451)

```
/*
 * Prototypes for table reading and writing functions.
 */
int fputints(const char *name, int *ptr, int max);
int fgetints(const char *name, int *ptr, int max);
```

getints1.c

(Figure 18.5 – Page 451)

```
/*
 * Reading and writing arrays using formatted I/O.
 *   fgetints - read values from file into an array.
 *   fputints - write values from array into a file.
 */
#include <stdio.h>
#include "getints.h"

int fgetints(const char *name, int a[], int max)
{
  int  cnt = 0;
  FILE *fp = fopen(name, "r");

  if (fp != NULL)
  {
    for (; cnt < max && fscanf(fp, "%i", &a[cnt]) == 1; cnt++)
        ;        /* read values while there's room */
    fclose(fp);
  }
  return cnt;    /* number of values actually read */
}

int fputints(const char *name, int a[], int num)
{
  int  cnt = 0;
  FILE *fp = fopen(name, "w");

  if (fp != NULL)
  {
    for (; cnt < num; cnt++)
      fprintf(fp, "%i\n", a[cnt]);
    fclose(fp);
  }
  return cnt;       /* assumes write successful */
}
```

filecpy2.c

(Figure 18.6 – Page 452)

```
/*
 * Copy one file into another, a line at a time.
 */
#include <stdio.h>
#include <stddef.h>
#include <string.h>

#define MAXLEN  80

long filecpy(const char *dest, const char *source)
{
  FILE *sfp, *dfp;              /* source and destination files */
  char line[MAXLEN + 1];        /* next input line */
  long cnt = -1L;               /* count of characters written */

  if ((sfp = fopen(source,"rb")) == NULL)
    printf("Can't open %s for reading\n", source);
  else
  {
    if ((dfp = fopen(dest,"wb")) == NULL)
      printf("Can't open %s for writing\n", dest);
    else
    {
      for (cnt = 0L; fgets(line, sizeof(line), sfp) != NULL;
           cnt += strlen(line))
        fputs(line,dfp);
      fclose(dfp);
    }
    fclose(sfp);
  }
  return cnt;
}
```

fileopen.h

(Figure 18.7 – Page 454)

```
/*
 * Prototypes for fileopen/fileclose functions.
 */
#include <stdio.h>
#include <stddef.h>

extern FILE *fileopen(const char *name, const char *mode);
extern void fileclose(const char *name, FILE *fp);
```

fileopen.c

(Figure 18.8 – Page 454)

```
/*
 * Open/close files, with error message on failure.
 *   fileopen - open a file (built on top of fopen).
 *   fileclose - close a file (built on top of fclose).
 */
#include "fileopen.h"

FILE *fileopen(const char *name, const char *mode)
{
  FILE *fp = fopen(name, mode);            /* open the file */

  if (fp == NULL)                          /* did open fail? */
  {
    fprintf(stderr, "Can't open %s for ", name);
    switch(mode[0])
    {
      case 'r': fprintf(stderr, "reading\n");
                break;
      case 'w': fprintf(stderr, "writing\n");
                break;
      case 'a': fprintf(stderr, "appending\n");
                break;
      default:  fprintf(stderr, "some strange mode\n");
                break;
    }
  }
  return fp;
}

void fileclose(const char *name, FILE *fp)
{
  if (fp != NULL && fclose(fp) == EOF)
    fprintf(stderr, "Error closing %s\n", name);
}
```

filelen.c

(Figure 18.9 – Page 456)

```
/*
 * Find file's length in bytes. Usage: filelen file1 ...
 */
#include <stdio.h>
#include "fileopen.h"

int main(int argc, char *argv[])
{
  long filelen(FILE *fp);
  int  i;                              /* argument length */
  long len;                            /* next file's length */
  FILE *fp;                            /* file pointer */

  for (i = 1; i < argc; i++)
  {
    if ((fp = fileopen(argv[i], "rb")) != NULL)
      if ((len = filelen(fp)) == -1)
        fprintf(stderr,"Can't compute length of %s\n", argv[i]);
      else
        printf("%s: %i\n", argv[i], len);
    fileclose(argv[i], fp);
  }
  return 0;
}

long filelen(FILE *fp)
{
  long oldpos = ftell(fp);              /* save old position */
  long length;

  if (fseek(fp, 0L, SEEK_END) == -1)    /* goto end of file */
    return -1;
  length = ftell(fp);                   /* compute length */
  return (fseek(fp, oldpos, SEEK_SET) == -1) ? -1 : length;
}
```

tail.c

(Figure 18.10 – Page 457)

```
/*
 * Print last 10 lines of a file.
 */
#include "fileopen.h"

#define LAST     10           /* last 10 lines of the file */

int main(int argc, char *argv[])
{
  long findpos(FILE *fp, int lastline);
  int  i;                     /* index */
  long pos;                   /* position of newline within block */
  FILE *fp;                   /* file we're reading through */
  int  c;                     /* next character to display */

  for (i = 1; i < argc; i++)
    if ((fp = fileopen(argv[i],"rb")) != NULL)
    {
      if ((pos = findpos(fp, LAST)) == -1L
           || fseek(fp, pos, SEEK_SET) == -1)
        fprintf(stderr,"Couldn't tail %s\n", argv[i]);
      else
        while ((c = getc(fp)) != EOF)    /* display lines */
          putc(c, stdout);
      fileclose(argv[1], fp);
    }
  return 0;                   /* no special return value for error */
}
```

tailutls.c

(Figure 18.11 – Page 458)

```
/*
 * Utility functions for tail program.
 *   findpos - locate first character in file to print.
 *   nextblock - read next block of characters in file.
 */
#include <stdio.h>

#define MAXBUF 512          /* characters in block */

long findpos(FILE *fp, int lines)
{
  int  nextblock(FILE *fp, int max, int *cnt, int target);
  int  count = 0;           /* newlines seen */
  long pos;                 /* position of desired newline in file */
  long blkend;              /* position of last char in block */
  int  blksize;             /* # of characters within block */

  if (fseek(fp, 0L, SEEK_END) == -1)
     return -1;             /* get to end of file */

  for (blkend = ftell(fp); blkend >= 0; blkend -= MAXBUF)
  {
    blksize = (MAXBUF > blkend) ? (int) blkend : MAXBUF;
    if (fseek(fp, blkend - blksize, SEEK_SET) == -1)
       return -1;           /* couldn't go back a block - error */
    if ((pos = nextblock(fp, blksize, &count, lines)) != -1)
       return blkend - blksize + pos;
  }

  return 0L;                /* start at beginning */
}
```

tailutls.c

(Figure 18.11 – Page 458)

```
int nextblock(FILE *fp, int max, int *cnt, int target)
{
  char buffer[MAXBUF];                /* to hold block */
  char *ptr    = buffer;              /* pointer into buffer */
  char *endptr = buffer + max;        /* pointer to end of buffer */
  int  c;

  for (; ptr < endptr && (c = getc(fp)) != EOF; *ptr++ = c)
    ;                                 /* read next block */
  *ptr = '\0';
  for (; ptr >= buffer; ptr--)
     if (*ptr == '\n' && (*cnt)++ == target)
       return ptr - buffer + 1;       /* where to start reading */
  return -1;                          /* have to read another block */
}
```

getints2.c

(Figure 18.12 – Page 459)

```
/*
 * New versions of our earlier functions to read and write arrays.
 * Now they use fread/fwrite to read/write the arrays all at once.
 */
#include "fileopen.h"
#include "getints.h"

int fgetints(const char *name, int a[], int max)
{
  FILE *fp = fileopen(name, "rb");
  int  cnt = -1;

  if (fp != NULL)
    cnt = fread((void *) a, sizeof(int), max, fp);
  fileclose(name, fp);
  return cnt;                       /* # of elements written or -1 */
}

int fputints(const char *name, int a[], int num)
{
  FILE *fp = fileopen(name, "wb");
  int  cnt = -1;

  if (fp != NULL)
    cnt = fwrite((void *) a, sizeof(int), num, fp);
  fileclose(name, fp);
  return cnt;                       /* # of elements written or -1 */
}
```

varray.h

(Figure 18.13 – Page 461)

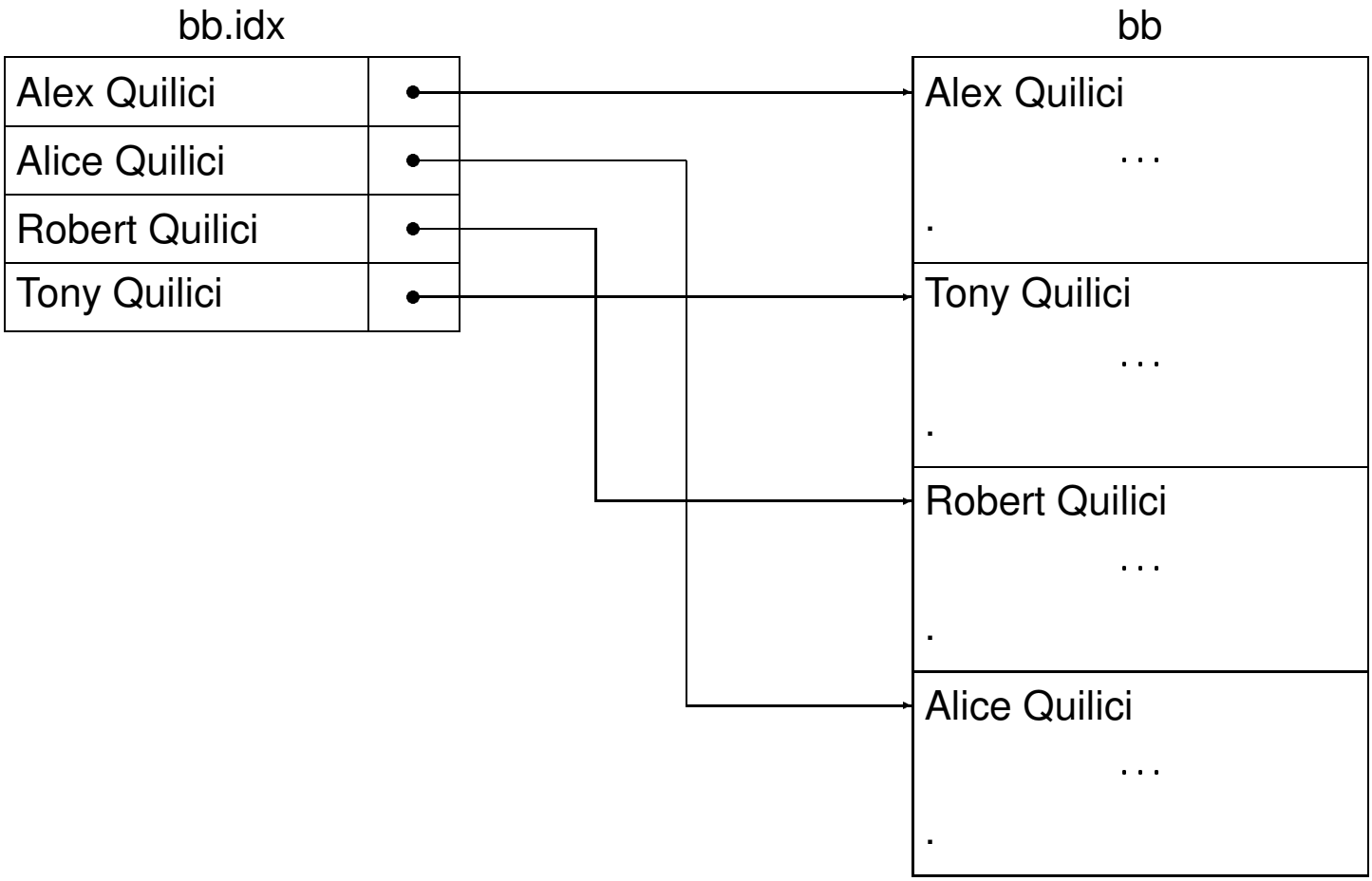
```
/*
 * Define virtual array type, prototypes, and macros.
 */
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>

typedef struct varray
{
  FILE *fp;      /* pointer to external file */
  int  size;     /* size of an element */
} VARRAY;

VARRAY *va_create(const char *name, int size);
VARRAY *va_open(const char *name, int size);

#define va_close(vaptr) ((void) fclose(vaptr->fp), free(vaptr))

#define va_entries(vaptr) \
  (fseek((vaptr)->fp, 0L, SEEK_END) == -1L \
      ? -1 : (int) (ftell((vaptr)->fp) / (vaptr)->size))

#define va_get(vaptr, i, ptr)  \
  (fseek((vaptr)->fp, (long) (i) * (vaptr)->size, SEEK_SET) != -1 && \
         fread((ptr), (vaptr)->size, 1, (vaptr)->fp) == 1)

#define va_put(vaptr, i, ptr) \
  (fseek((vaptr)->fp, (long) (i) * (vaptr)->size, SEEK_SET) != -1 && \
         fwrite((ptr), (vaptr)->size, 1, (vaptr)->fp) == 1)
```

varray.c

(Figure 18.14 – Page 462)

```
/*
 * Operations for creating or opening a virtual array.
 */
#include "allocs.h"               /* Our memory macros (CHAP15) */
#include "varray.h"

static VARRAY *make_varray(const char *name, const char *mode, int size)
{
  FILE   *fp  = fopen(name, mode);
  VARRAY *new = (fp != NULL) ? ALLOC(1, VARRAY) : NULL;

  if (new != NULL)
  {
    new->fp = fp;
    new->size = size;
  }
  else if (fp == NULL)
    (void) fclose(fp);
  return new;
}

VARRAY *va_create(const char *name, int size)  /* create a new VARRAY */
{
  return make_varray(name, "w+b", size);
}

VARRAY *va_open(const char *name, int size)    /* open existing VARRAY */
{
  return make_varray(name, "r+b", size);
}
```

ADDRESS BOOK ORGANIZATION



index.h

(Figure 18.16 – Page 464)

```
/*
 * Definitions to use indexes.
 */
#include <string.h>
#include "varray.h"                     /* for external array */

#define MAXLEN     80                   /* maximum line length */
#define MAXKEY     40                   /* maximum key size */
#define INDEXFILE  "idx"                /* the index file */
#define ENDREC(x)  ((*x) == '.')        /* end of a record */
#define MAXIDX     100                  /* max number of records */

struct indexrec                         /* index record: */
{
  char key[MAXKEY];                     /*   name */
  long pos;                             /*   position */
};

typedef struct indexrec INDEX;
```

index.c

(Figure 18.17 – Page 465)

```
/*
 * Create the index table for a file.
 */
#include "index.h"                      /* definition of INDEX */
#include "fileopen.h"

int main(int argc, char *argv[])
{
  int   maketable(FILE *fp, INDEX table[]);
  int   savetable(char *name, INDEX table[], int n);
  int   cmpkey(const void *, const void *);
  INDEX itable[MAXIDX];                 /* holds index table */
  int   items;                          /* items in table */
  char  iname[MAXLEN + 1];              /* index file name */
  FILE  *fp;                            /* text file name */

  if (argc != 2)
  {
    fprintf(stderr,"Usage: %s file\n", argv[0]);
    return 1;                           /* exit early, bad arguments */
  }
```

index.c

(Figure 18.17 – Page 465)

```
  if ((fp = fileopen(argv[1], "rb")) == NULL)
    return 1;                          /* exit early, file problems */
  if ((items = maketable(fp, itable)) == -1)
  {
    fprintf(stderr,"Too many index entries\n");
    return 1;                          /* exit early, index problems */
  }
  qsort(itable, items, sizeof(INDEX), cmpkey);
  sprintf(iname,"%s.%s", argv[1], INDEXFILE);
  if (savetable(iname, itable, items) != items)
  {
    fprintf(stderr, "Couldn't build index file\n");
    return 1;                          /* exit early, index problems */
  }
  fileclose(argv[1], fp);

  return 0;
}

int cmpkey(const void *xptr, const void *yptr)
{
  return strcmp(((INDEX *) xptr)->key, ((INDEX *) yptr)->key);
}
```

indextab.c

(Figure 18.18 – Page 466)

```
/*
 * Functions to construct and save the index table.
 *   savetable - write index table into file.
 *   maketable - build index table from reading file.
 */
#include "index.h"

int savetable(char *name, INDEX table[], int n)
{
  int i = -1;                 /* indicates create failed error */
  VARRAY *vaptr;              /* the virtual array we create */

  if ((vaptr = va_create(name, sizeof(INDEX))) != NULL)
  {
    for (i = 0; i < n; i++)
      if (va_put(vaptr, i, &table[i]) == -1)
        break;
    va_close(vaptr);
  }
  return i;
}
```

indextab.c

(Figure 18.18 – Page 466)

```
int maketable(FILE *fp, INDEX table[])
{
  char line[MAXLEN + 1];      /* next line */
  long pos;                   /* record's pos in file */
  int  linecnt = 0;           /* # of lines in record */
  int  cnt = 0;               /* count of records */
  int  last;                  /* index to last character in line */

  while (pos = ftell(fp), fgets(line,sizeof(line), fp) != NULL)
  {
    if (linecnt++ == 0)       /* add a table entry */
    {
      if (cnt == MAXIDX)
        return -1;
      last = strlen(line) - 1;          /* strip non-key chars */
      if (line[last - 1] == '\r')       /* \r\n ends line */
        line[last - 1] = '\0';
      else                              /* \n ends line */
        line[last] = '\0';
      strncpy(table[cnt].key, line, MAXKEY);
      table[cnt++].pos = pos;           /* store position */
    }
    if (ENDREC(line))
      linecnt = 0;
  }
  return cnt;
}
```

lookup.c

(Figure 18.19 – Page 467)

```
/*
 * Look up addresses/phone numbers given name.
 */
#include "index.h"
#include "fileopen.h"

int main(int argc, char *argv[])
{
  int  findnames(FILE *fp, char *name);
  FILE *fp;                              /* address book */
  char iname[MAXLEN + 1];                /* index name */
  int  status = 1;                       /* return value */

  if (argc != 2)
    fprintf(stderr, "usage: %s file\n", argv[0]);
  else if ((fp = fileopen(argv[1], "rb")) != NULL)
  {
    sprintf(iname, "%s.%s", argv[1], INDEXFILE);
    if (!findnames(fp, iname))
    {
      fileclose(argv[1],fp);
      status = 0;
    }
  }
  return status;
}
```

lookup.c

(Figure 18.19 – Page 467)

```
int findnames(FILE *fp, char *name)
{
  long   va_search(VARRAY *ep, int first, int last, char *target);
  long   display_record(FILE *fp, long pos);
  int    getline(char *, int);
  int    entries;                        /* # of entries in file */
  VARRAY *vaptr;                         /* virtual array */
  char   line[MAXLEN + 1];               /* user input line */
  long   pos;                            /* position of record */

  if ((vaptr = va_open(name, sizeof(INDEX))) == NULL)
  {
    fprintf(stderr,"Can't access index file %s\n", name);
    return 0;   /* oops... */
  }
  entries = va_entries(vaptr);
  while (fputs("Name? ", stdout), getline(line, MAXLEN) != -1)
    if ((pos = va_search(vaptr, 0, entries - 1, line)) == -1)
      fprintf(stderr, "Couldn't find: %s\n", line);
    else if (display_record(fp, pos) == -1)
      fprintf(stderr, "Couldn't display record at %li\n", pos);
  va_close(vaptr);
  return 1;     /* success */
}
```

find.c

(Figure 18.20 – Page 468)

```
/*
 * Get name from user, look it up, and display record.
 */
#include <string.h>
#include "index.h"

long display_record(FILE *fp, long pos)
{
  int  rval;                          /* fseek's return value */
  char line[MAXLEN + 1];              /* holds user input */

  if ((rval = fseek(fp, pos, SEEK_SET)) != -1)
    while (fgets(line, sizeof(line), fp) && !ENDREC(line))
      fputs(line,stdout);
  return rval;
}

long va_search(VARRAY *vaptr, int first, int last, char *target)
{
  int   mid = (first + last) / 2;     /* mid item */
  INDEX next;                         /* to hold record */
  int   cmp;                          /* holds comparison result */

  if (last < first || !va_get(vaptr, mid, (void *) &next))
    return -1L;                       /* not there */
  if ((cmp = strncmp(target, next.key, MAXKEY)) == 0)
    return next.pos;                  /* found it */
  return (cmp < 0)  ? va_search(vaptr, first, mid - 1, target)
                    : va_search(vaptr, mid + 1, last, target);
}
```

REMINDERS

- Don't open a file for writing unless you're sure you no longer need the data it contains.

- **`fread`** and **`fwrite`** are portable, but the files they read and write aren't.

CHAPTER SUMMARY

- C allows us to access external files through library functions.

- We use **`fopen`** to open files and **`fclose`** to close them. To open a file, we need to specify the file name and how we're going to use it. The three files **`stdin`**, **`stdout`**, and **`stderr`** are automatically opened for us.

- We can use **`getc`** and **`putc`** to do character-at-a-time I/O to files. **`getchar`** and **`putchar`** are macros defined on top of these functions.

- We can use **`fgets`** and **`fputs`** to do line-at-a-time I/O to files. These usually are slower than **`getc`** and **`putc`**, and they aren't designed to be easy to use, so we tend not to use them very much.

CHAPTER SUMMARY

- We can use **`fseek`** to move to any byte in a file, **`rewind`** to move to the beginning of a file, and **`ftell`** to find out where in the file we actually are.

- We use **`fread`** and **`fwrite`** to read and write blocks of bytes to a file.

- There are special update modes, indicated with a “+” sign, that allow us to both read and write the same file.