Programming with C

Terry Marris December 2010

19 Binary Files

In the previous chapter we looked at text files. Text files contain variable length records implemented as lines of text. Now we look at binary files.

19.1 Records and Fields

Binary files contain fixed length records implemented as structures.

```
typedef struct tagCatalogueRecord {
  char author[25];  /* indexed 0..24 */
  char title[25];
  int nCopies;
} CatalogueRecord;
```

A structure's members are known as *fields*.

newRecord() creates a new record from the given fields.

```
/* newRecord: returns a new record from the given
   author, title and # copies */
CatalogueRecord newRecord(char *author, char *title, int nCopies)
{
  CatalogueRecord record;

  strncpy(record.author, author, 24);
  record.author[24] = '\0';
  strncpy(record.title, title, 24);
  record.title[24] = '\0';
  record.nCopies = nCopies;
  return record;
}
```

strncpy(string1, n, string2) copies at most *n* characters from *string2* to *string1*. If exactly *n* characters are copied, the end-of-string null character is not copied; we have to explicitly copy it to *string1* ourselves.

printRecord() displays the fields in a record.

```
/* printRecord: prints a single record */
int printRecord(CatalogueRecord record)
{
  printf("%-25s %-25s %d\n",
            record.author, record.title, record.nCopies);
  return 0;
}
```

19.2 Create Binary File

To create a binary file we open it in write binary mode, write records to it, and then close the file.

```
/* createFile: creates a binary file */
int createFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  catalogue = fopen(fileName, "wb");

  record = newRecord(
            "Kernighan & Ritchie", "The C Programming Language", 10);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  ...
  fclose(catalogue);
  return 0;
}
```

The line

```
FILE *catalogue;
```

declares *catalogue* to be of type pointer to *FILE*. *FILE* is defined in *stdio.h*.

We make the connection between the internal (to the program) and external filenames with

```
catalogue = fopen(fileName, "wb");
```

fopen() opens the file and *wb* specifies open the file in write binary mode. If the file already exists on disk, it is overwritten with the new file. *fopen()* and *wb* are both defined in *stdio.h*.

We write a record to the file with

```
fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
```

The statement says write 1 copy of the data held in *record*, which has a size determined by *sizeof*, to the *catalogue* file.

Finally, we close the file.

```
fclose(catalogue);
```

Closing the file breaks the connection between the internal and external filenames, and flushes file buffers so that all the data is physically written to the file.

19.3 Print Binary File

To print the contents of a binary file we first open the file for reading, retrieve records and print them until the end of the file is reached. Then, finally, we close the file.

```
/* printFile: displays the contents of the given file */
int printFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  printHeadings();
  catalogue = fopen(fileName, "rb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    printRecord(record);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  return 0;
}
```

We declare the file and record variables with

```
FILE *catalogue;
CatalogueRecord record;
```

The file is opened in read binary mode.

```
  catalogue = fopen(fileName, "rb");
```

We do a priming read.

```
fread(&record, sizeof(CatalogueRecord), 1, catalogue);
```

This retrieves the first record and sets *feof()*, *feof()* for end-of-file.

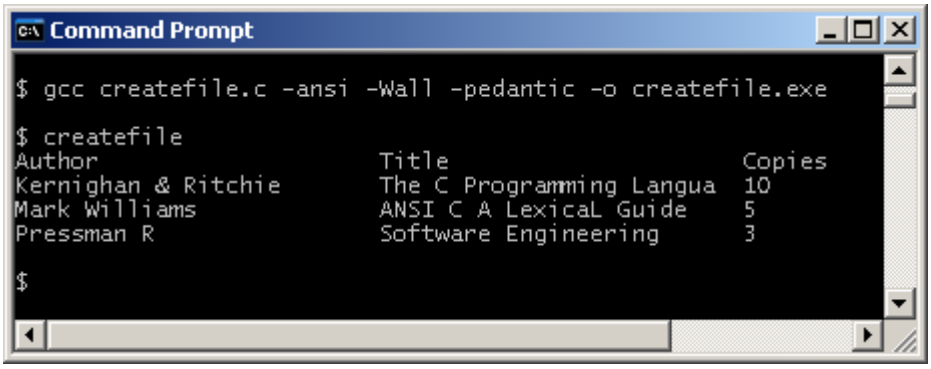
If *feof()* is false we go into the loop, print the record just retrieved, and then attempt to retrieve the next record. If f*eof()* is true we exit the loop and close the file.

Here is the complete program and its run.

```
/* createfile.c: creates and prints a binary file */

#include <stdio.h>
#include <string.h>

typedef struct tagCatalogueRecord {
  char author[25];  /* indexed 0..24 */
  char title[25];
  int nCopies;
} CatalogueRecord;
```

```
/* newRecord: returns a new record from the given
   author, title and # copies */
CatalogueRecord newRecord(char *author, char *title, int nCopies)
{
  CatalogueRecord record;

  strncpy(record.author, author, 24);
  record.author[24] = '\0';
  strncpy(record.title, title, 24);
  record.title[24] = '\0';
  record.nCopies = nCopies;
  return record;
}

/* printHeadings: prints headings */
int printHeadings()
{
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";
  return printf("%-25s %-25s %-6s\n", author, title, copies);
}

/* printRecord: prints a single record */
int printRecord(CatalogueRecord record)
{
  printf("%-25s %-25s %d\n",
            record.author, record.title, record.nCopies);
  return 0;
}

/* createFile: creates a binary file */
int createFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  catalogue = fopen(fileName, "wb");

  record = newRecord(
            "Kernighan & Ritchie", "The C Programming Language", 10);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  record = newRecord("Mark Williams", "ANSI C A LexicaL Guide", 5);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  record = newRecord("Pressman R", "Software Engineering", 3);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);

  fclose(catalogue);
  return 0;
}
```

```
/* printFile: displays the contents of the given file */
int printFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  printHeadings();
  catalogue = fopen(fileName, "rb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    printRecord(record);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  return 0;
}

int main()
{
  createFile("cat.data");
  printFile("cat.data");
  return 0;
}
```



Notice how the title *The C Programming Language* has been truncated. This is a consequence of having fixed length records. Declare them too small and data is lost. Declare them too large and space is wasted.

19.4 Append Records

To add records onto the end of the file we open the file in append binary mode.

```
catalogue = fopen(fileName, "ab");
```

Then we write records to the file.

```
record = newRecord("Deitel & Deitel", "C How to Program", 2);
fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
```

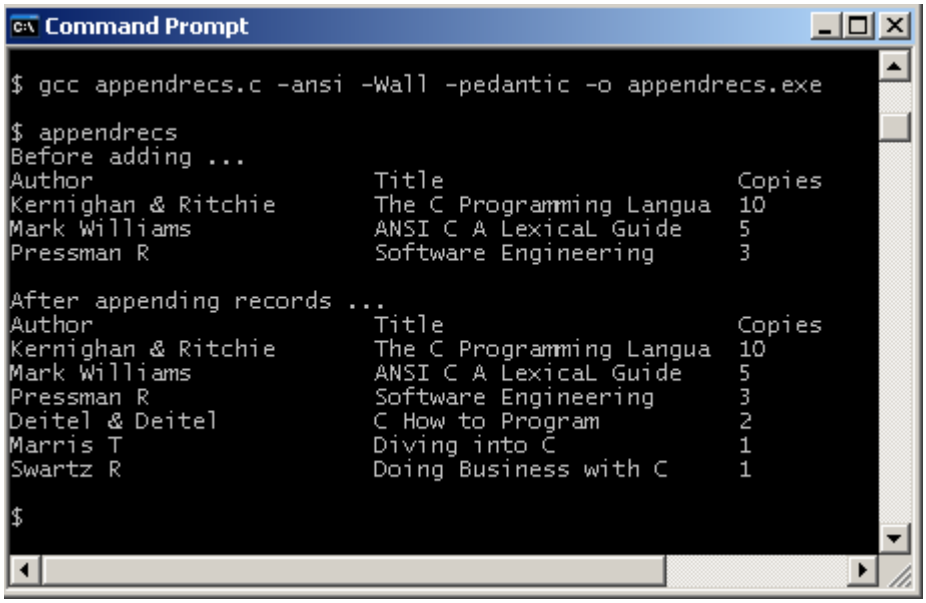
And we close the file when finished. Here is the entire function.

```
/* addRecords: adds records onto the end of an existing binary file
*/
int addRecords(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  catalogue = fopen(fileName, "ab");

  record = newRecord("Deitel & Deitel", "C How to Program", 2);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  record = newRecord("Marris T", "Diving into C", 1);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  record = newRecord("Swartz R", "Doing Business with C", 1);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);

  fclose(catalogue);
  return 0;
}
```

The entire program and its run are shown below.

```
/* appendrecs.c: adds records onto the end of a binary file */

#include <stdio.h>
#include <string.h>


typedef struct tagCatalogueRecord {
  char author[25];  /* indexed 0..24 */
  char title[25];
  int nCopies;
} CatalogueRecord;

/* newRecord: returns a new record from the given
   author, title and # copies */
CatalogueRecord newRecord(char *author, char *title, int nCopies)
{
  CatalogueRecord record;

  strncpy(record.author, author, 24);
  record.author[24] = '\0';
  strncpy(record.title, title, 24);
  record.title[24] = '\0';
  record.nCopies = nCopies;
  return record;
}

/* printHeadings: prints headings */
int printHeadings()
{
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";
  return printf("%-25s %-25s %-6s\n", author, title, copies);
}
```

```
/* printRecord: prints a single record */
int printRecord(CatalogueRecord record)
{
  printf("%-25s %-25s %d\n",
            record.author, record.title, record.nCopies);
  return 0;
}

/* addRecords: adds records onto the end of an existing binary file
*/
int addRecords(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  catalogue = fopen(fileName, "ab");

  record = newRecord("Deitel & Deitel", "C How to Program", 2);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  record = newRecord("Marris T", "Diving into C", 1);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
  record = newRecord("Swartz R", "Doing Business with C", 1);
  fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);

  fclose(catalogue);
  return 0;
}

/* printFile: displays the contents of the given file */
int printFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  printHeadings();
  catalogue = fopen(fileName, "rb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    printRecord(record);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  return 0;
}

int main()
{
  printf("Before adding ... \n");
  printFile("cat.data");
  addRecords("cat.data");
  printf("\nAfter appending records ... \n");
  printFile("cat.data");
  return 0;
}
```



19.6 Duplicate a Binary File

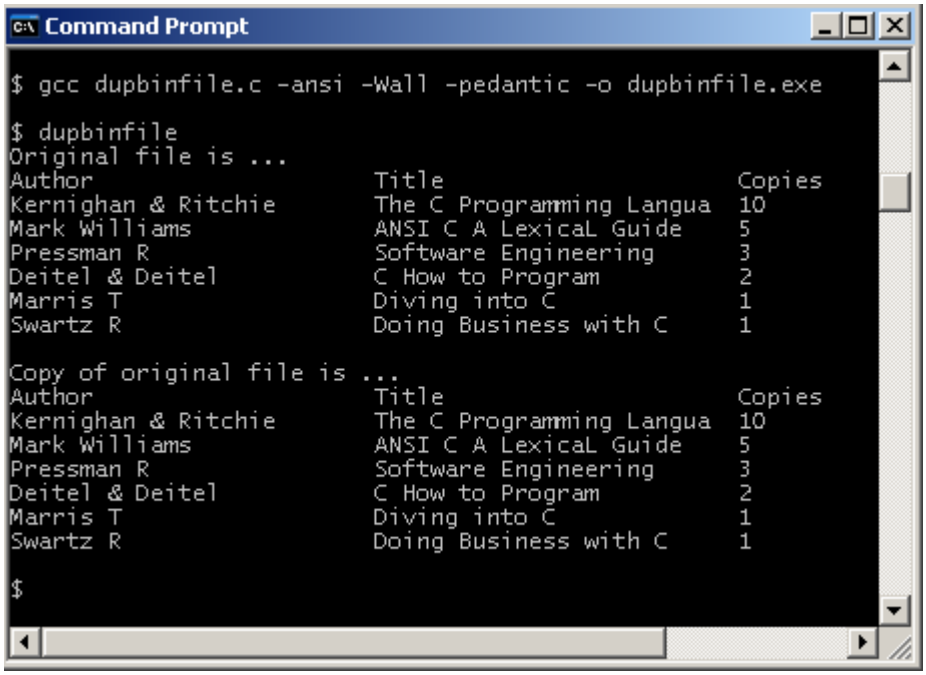
One way to create a copy of a file is to copy it record by record to a new file.

```
/* duplicateFile: creates a duplicate copy of a binary file */
int duplicateFile(char *fromFileName, char *toFileName)
{
  FILE *catalogue, *copy;
  CatalogueRecord record;

  catalogue = fopen(fromFileName, "rb");
  copy = fopen(toFileName, "wb");

  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    fwrite(&record, sizeof(CatalogueRecord), 1, copy);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }

  fclose(catalogue);
  fclose(copy);
  return 0;
}
```

The entire program and a run are shown below.

```
/* dupbinfile.c: creates a duplicate copy of a binary file */

#include <stdio.h>

typedef struct tagCatalogueRecord {
  char author[25];  /* indexed 0..24 */
  char title[25];
  int nCopies;
} CatalogueRecord;
```

```
/* printHeadings: prints headings */
int printHeadings()
{
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";
  return printf("%-25s %-25s %-6s\n", author, title, copies);
}

/* printRecord: prints a single record */
int printRecord(CatalogueRecord record)
{
  printf("%-25s %-25s %d\n",
            record.author, record.title, record.nCopies);
  return 0;
}

/* printFile: displays the contents of the given file */
int printFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  printHeadings();
  catalogue = fopen(fileName, "rb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    printRecord(record);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  return 0;
}

/* duplicateFile: creates a duplicate copy of a binary file */
int duplicateFile(char *fromFileName, char *toFileName)
{
  FILE *catalogue, *copy;
  CatalogueRecord record;

  catalogue = fopen(fromFileName, "rb");
  copy = fopen(toFileName, "wb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    fwrite(&record, sizeof(CatalogueRecord), 1, copy);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  fclose(copy);
  return 0;
}

int main()
{
  printf("Original file is ... \n");
  printFile("cat.data");
  duplicateFile("cat.data", "copyOfCat.data");
  printf("\nCopy of original file is ... \n");
  printFile("copyOfCat.data");
  return 0;
}
```



19.7 Remove a Record from a Binary File

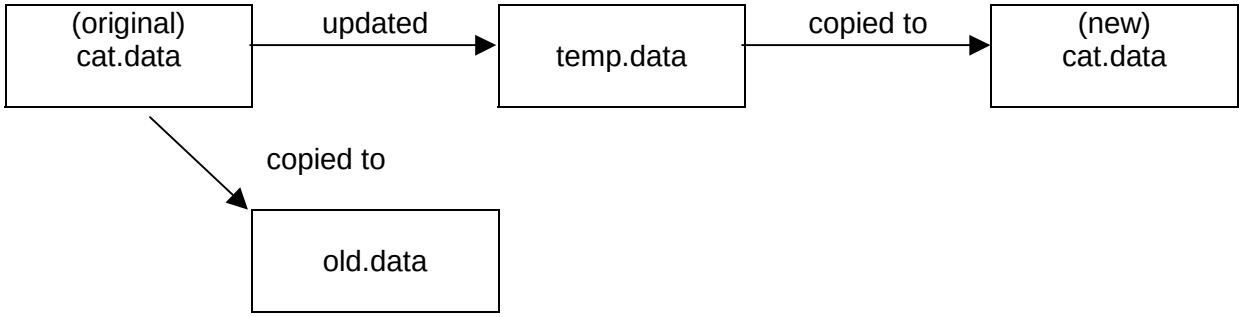
To remove a record from a file we copy its records, except the one to be deleted, to a new file.

```
/* deleteRecord: removes the record with the given author's name */
int deleteRecord(char *fileName, char *author)
{
  FILE *catalogue, *copy;
  CatalogueRecord record;

  catalogue = fopen(fileName, "rb");
  copy = fopen("temp.data", "wb");

  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    if (strcmp(record.author, author) != 0)
      fwrite(&record, sizeof(CatalogueRecord), 1, copy);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }

  fclose(catalogue);
  fclose(copy);

  duplicateFile(fileName, "old.data");
  duplicateFile("temp.data", fileName);

  return 0;
}
```

The statements

```
duplicateFile(fileName, "old.data");
duplicateFile("temp.data", fileName);
```

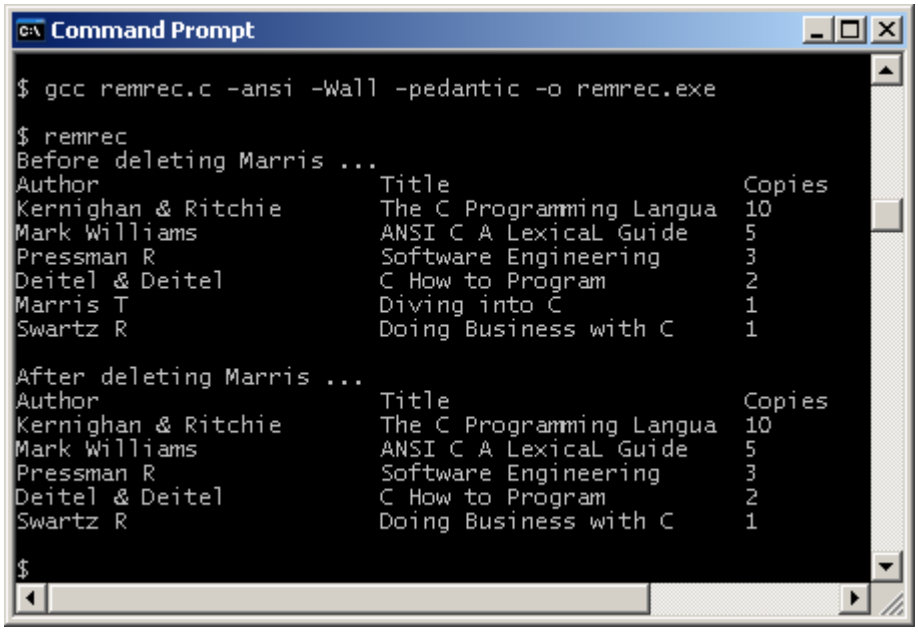
ensure the updated file has the same name as the original, and the original file is preserved in *old.data*.



The entire program and its run are shown below.

```
/* remrec.c: removes a record from a binary file */

#include <stdio.h>
#include <string.h>

typedef struct tagCatalogueRecord {
  char author[25];  /* indexed 0..24 */
  char title[25];
  int nCopies;
} CatalogueRecord;

/* printHeadings: prints headings */
int printHeadings()
{
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";
  return printf("%-25s %-25s %-6s\n", author, title, copies);
}

/* printRecord: prints a single record */
int printRecord(CatalogueRecord record)
{
  printf("%-25s %-25s %d\n",
            record.author, record.title, record.nCopies);
  return 0;
}

/* printFile: displays the contents of the given file */
int printFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;
  printHeadings();
  catalogue = fopen(fileName, "rb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    printRecord(record);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  return 0;
}
```

```
/* duplicateFile: creates a duplicate copy of a binary file */
int duplicateFile(char *fromFileName, char *toFileName)
{
  FILE *catalogue, *copy;
  CatalogueRecord record;

  catalogue = fopen(fromFileName, "rb");
  copy = fopen(toFileName, "wb");

  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    fwrite(&record, sizeof(CatalogueRecord), 1, copy);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }

  fclose(catalogue);
  fclose(copy);
  return 0;
}

/* deleteRecord: removes the record with the given author's name */
int deleteRecord(char *fileName, char *author)
{
  FILE *catalogue, *copy;
  CatalogueRecord record;

  catalogue = fopen(fileName, "rb");
  copy = fopen("temp.data", "wb");

  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    if (strcmp(record.author, author) != 0)
      fwrite(&record, sizeof(CatalogueRecord), 1, copy);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }

  fclose(catalogue);
  fclose(copy);

  duplicateFile(fileName, "old.data");
  duplicateFile("temp.data", fileName);

  return 0;
}

int main()
{
  printf("Before deleting Marris ... \n");
  printFile("cat.data");
  deleteRecord("cat.data", "Marris T");
  printf("\nAfter deleting Marris ... \n");
  printFile("cat.data");
  return 0;
}
```



Another way to remove a record from a file, and perhaps the best way, is to have a field, named *status* say, with values either *current* or *deleted*. To remove a record we just change its *status* from *current* to *deleted*. OK. The record physically stays in the file. But it is logically deleted. You can refer back to the logically deleted record some time in the future if you need to.

19.8 Amend a Record in a Binary File

ANSI C A Lexical Guide is such a useful book that the library has acquired two more copies. We need to amend its record to match.

C maintains a record number, starting with zero, for records as structures written to a file. Each record has its own unique identifying record number. We make use of this

We open the file in update binary mode.

```
catalogue = fopen(fileName, "rb+");
```

We read through the file, keeping a count of records read as we do so, until we find the record that need updating.

```
int recNumber = 0;
fread(&record, sizeof(CatalogueRecord), 1, catalogue);
while (!feof(catalogue)) {
  if (strcmp(record.title, title) == 0) {
    ...
  }
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  recNumber++;
}
```

If we find the required record , we update its number of copies, use *fseek()* to reposition the file position indicator, then write the amended record to the file.

```
record.nCopies = record.nCopies + copies;
fseek(catalogue, recNumber * sizeof(CatalogueRecord),SEEK_SET);
fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
```

The file position indicator is the point where the next read or write operation will occur in the file. *SEEK_SET* means *from the beginning of the file*. So, the statement

```
fseek(catalogue, recNumber * sizeof(CatalogueRecord),SEEK_SET);
```

moves the file position indicator to *recNumber * sizeof(CatalogueRecord)* bytes from the beginning of the *catalogue* file.

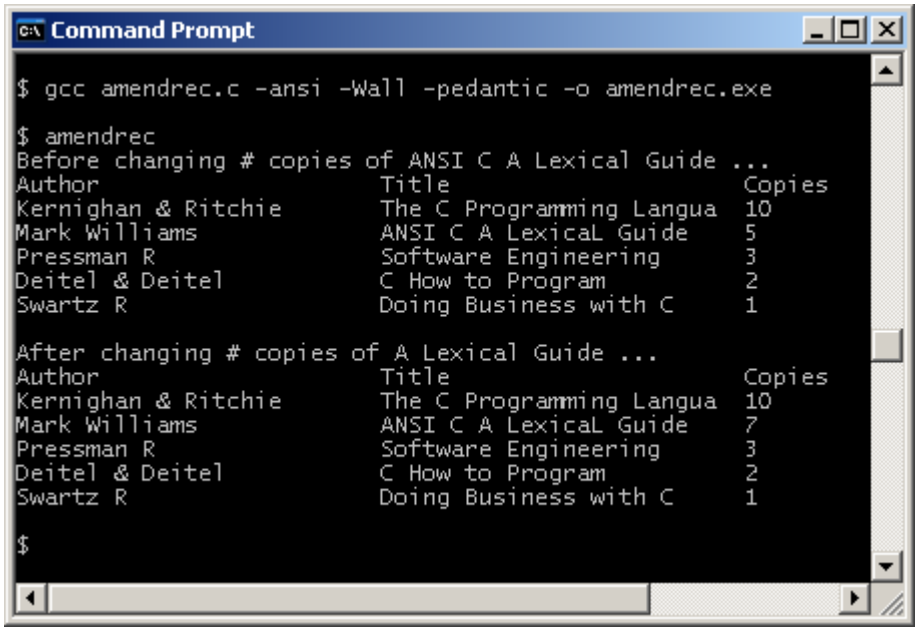
Here is the complete function.

```
/* amendRecord: adds the given number of copies to the record with
   the given title */
int amendRecord(char *fileName, char *title, int copies)
{
  FILE *catalogue;
  CatalogueRecord record;
  int recNumber = 0;

  catalogue = fopen(fileName, "rb+");

  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    if (strcmp(record.title, title) == 0) {
      record.nCopies = record.nCopies + copies;
      fseek(catalogue, recNumber * sizeof(CatalogueRecord),SEEK_SET);
      fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
      break;
    }
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
    recNumber++;
  }
  fclose(catalogue);
  return 0;
}
```

The entire program and its run are shown below.

```
/* amendrec.c: changes number of copies for
   a record identified by its title */
#include <stdio.h>
#include <string.h>

typedef struct tagCatalogueRecord {
  char author[25];  /* indexed 0..24 */
  char title[25];
  int nCopies;
} CatalogueRecord;

/* printHeadings: prints headings */
int printHeadings()
{
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";
  return printf("%-25s %-25s %-6s\n", author, title, copies);
}
```

```
/* printRecord: prints a single record */
int printRecord(CatalogueRecord record)
{
  printf("%-25s %-25s %d\n",
            record.author, record.title, record.nCopies);
  return 0;
}

/* printFile: displays the contents of the given file */
int printFile(char *fileName)
{
  FILE *catalogue;
  CatalogueRecord record;

  printHeadings();
  catalogue = fopen(fileName, "rb");
  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    printRecord(record);
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  }
  fclose(catalogue);
  return 0;
}

/* amendRecord: adds the given number of copies to the record with
   the given title */
int amendRecord(char *fileName, char *title, int copies)
{
  FILE *catalogue;
  CatalogueRecord record;
  int recNumber = 0;

  catalogue = fopen(fileName, "rb+");

  fread(&record, sizeof(CatalogueRecord), 1, catalogue);
  while (!feof(catalogue)) {
    if (strcmp(record.title, title) == 0) {
      record.nCopies = record.nCopies + copies;
      fseek(catalogue, recNumber * sizeof(CatalogueRecord),SEEK_SET);
      fwrite(&record, sizeof(CatalogueRecord), 1, catalogue);
      break;
    }
    fread(&record, sizeof(CatalogueRecord), 1, catalogue);
    recNumber++;
  }
  fclose(catalogue);
  return 0;
}

int main()
{
  printf("Before changing #copies of ANSI C A Lexical Guide ... \n");
  printFile("cat.data");
  amendRecord("cat.data", "ANSI C A LexicaL Guide", 2);
  printf("\nAfter changing # copies of A Lexical Guide ... \n");
  printFile("cat.data");
  return 0;
}
```



We can see that the number of copies for Mark Williams ANSI C A LexicaL Guide has changed from 5 to 7.

19.9 What can go Wrong?

What can go wrong is always an important question to ask ... and to answer.

A file opened in a read mode must exist. If the file cannot be found *fopen()* returns *NULL*.

```
FILE *file;
int report = fopen(fileName, "rb");
if (report == NULL)
  printf("Error.  %s cannot be opened for reading\n", fileName);
```

fread() or *fwrite()* will fail if the storage medium unexpectedly becomes offline, by removing a memory stick for example. *fwrite()* will also fail if the storage medium becomes full, or the user does not have write permission.

fread() returns the number of items read and *fwrite()* returns the number of items written. For example in

```
fread(&record, sizeof(CatalogueRecord), 1, catalogue);
```

fread() attempts to read one item.

```
int report = fread(&record, sizeof(CatalogueRecord), 1, catalogue);
if (report != 1)
  printf("Error in %s.  Failed to read a record");
```

fseek() returns a non-zero value if you attempt to seek past the end of a file, or if you attempt an *fseek()* on a file that does not exist.

```
int report = fseek(catalogue,recNumber * sizeof(CatalogueRecord),SEEK_SET);
if (report != 0)
  printf("Error.  Attempt to seek past the end of %s", fileName);
```

You may have noticed that several different programs share the same record structure. You could save just one copy of the structure in a header file, and *#include* it in the programs that require it; this ensures consistency between programs. You could also do the same with external file names.

You may have noticed that the same functions are used in different programs, *printRecord()* for example. You could also include them in a header file. Or you could compile them to create an object file, and then link the object file with different programs as required This technique is discussed in the chapter on Modules.

Exercise 19.1

1. A student has a unique five-digit reference number, a name, and a number of credits obtained by completing assignments. Create a binary file of student records. Print the contents of the file. Note: you do not perform arithmetic with student reference numbers, so it makes sense to declare this number as an array of *char*. Include as much error checking as you reasonably can.

2. Write and test a function that will delete a student's record from the file created in question 1 above. Remember to print the contents of the file before and again after the deletion and to include as much error checking as you reasonably can.

3. A student has passed another assignment and, so, their record needs updating. Write and test a function that will update a student's record. Remember to print the contents of the file both before and after the update and to include as much error checking as you reasonably can.

We have looked at text files. We have looked at binary files. **Next** we take a look at date and time functions.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988
Deitel H and Deitel P *C How to Program* Prentice Hall 1992 pp 357