Programming with C

Terry Marris December 2010

18 Text Files

In the previous chapter we looked at structures and arrays of structures. Now we look at files of text.

An array has its size fixed by the programmer at compile time. The user might find this size too limiting. Or the programmer might be too generous and waste a finite resource. Further, the contents of the array are lost when the program closes down. We need the user to decide on storage size (within physical limits). And we need the data to be preserved between program runs. We need files.

We focus first on the straightforward cases, then, in the next chapter, investigate what can go wrong .

18.1 Records and Fields

A text file contains *lines* of text. Each line is known as a *record*. A record is split into components known as *fields*.

catalogue

Kernighan & Ritchie, The C Programming Language, 10
Mark Williams, ANSI C A Lexical Guide, 5
Pressman R, Software Engineering, 3

In this example, each record has three fields: author, title and number of copies held. A comma and a space together are used to separate each field from the next. *catalogue* is the file name.

18.2 Lines

We assemble a line of text from its components, then dis-assemble it and print each component.

In assembling a line we convert *int*'s to strings, compute the size of each component, add an allowance for separators such as commas and for the null end-of-string character, then use *strcpy()* to copy the first component to the line, and *strcat()* to add subsequent components.

We remember that *intToString()* was developed in the chapter on Arrays of Char and included in *utility.c*.

```
/* newRecord: assembles a new line from
   the given author, title and number of copies */
char *newRecord(char *author, char *title, int nCopies)
{
  int size;
  char *line;
  char *copies = intToString(nCopies);

  size = strlen(author) + strlen(title) + strlen(copies);
  size = size + 3; /* +3 for , , and null */

  line = malloc(size);
  strcpy(line, author);
  strcat(line, ",");
  strcat(line, title);
  strcat(line, ",");
  strcat(line, copies);
  return line;
}
```

We dis-assemble the line using *strtok()*. *strtok()* splits a line up into *tokens* e.g. words separated by commas.

The first call to *strtok()* receives the line to be split, along with a string of separators, and returns the first token. Our separator string contains just one character, the comma (but you could have others such as space as well as a comma). The second and subsequent calls to *strtok()* receive *NULL* and a string of separators, and each call returns the next token. *strtok()* returns *NULL* if it cannot find any more tokens.

```
/* printRecord: splits line into components and prints them */
int printRecord(char *line)
{
  char *ptr;

  for (ptr = strtok(line, ","); ptr != NULL; ptr = strtok(NULL, ","))
    printf("%s  ", ptr);
  return printf("\n");
}
```

The problem here is that no consideration is given to individual field widths. So the data items cannot be listed neatly under their column headings. We attempt to address this problem in the next version of *printRecord()*.

In this version we store each token returned by *strtok()* in an array, then we can apply suitable formatting to each element of the array when printed. Of course we assume that *strtok()* returns the right number of tokens to give each element of the array a value.

```
/* printRecord: splits line into components and prints them */
int printRecord(char *line)
{
  char *ptr;
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";

  char* array[3];  /* indexed 0..2 */
  int i;

  for (ptr = strtok(line, ","), i = 0;
      ptr != NULL && i < 3;
      ptr = strtok(NULL, ","), i++)
    array[i] = ptr;

  printf("%-25s %-25s %-10s\n", author, title, copies);
  printf("%-25s %-25s %-10s\n", array[0], array[1], array[2]);
  return printf("\n");
}
```

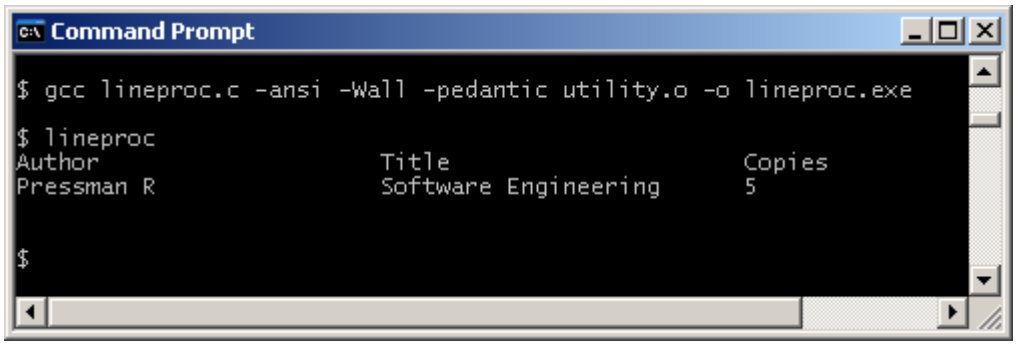
Check out the *for* loop. The initialisation stage contains two expressions separated by a comma, as does the conditional stage, as does the re-initialisation stage.

newRecord() and *printRecord()* are tightly coupled in that what is to be displayed relies on knowing what is saved by *newRecord()*.

Here is the complete program and its run.

```
/* lineproc.c: assembles a line from given strings, then
               dis-assembles the line and prints the components */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include "utility.h"

/* newRecord: assembles a new line from
   the given author, title and number of copies */
char *newRecord(char *author, char *title, int nCopies)
{
  int size;
  char *line;
  char *copies = intToString(nCopies);

  size = strlen(author) + strlen(title) + strlen(copies);
  size = size + 3; /* +3 for , , and null */

  line = malloc(size);
  strcpy(line, author);
  strcat(line, ",");
  strcat(line, title);
  strcat(line, ",");
  strcat(line, copies);
  return line;
}
```

```
/* printRecord: splits line into components and prints them */
int printRecord(char *line)
{
  char *ptr;
  char *author = "Author";
  char *title = "Title";
  char *copies = "Copies";

  char* array[3];  /* indexed 0..2 */
  int i;

  for (ptr = strtok(line, ","), i = 0;
      ptr != NULL && i < 3;
      ptr = strtok(NULL, ","), i++)
    array[i] = ptr;

  printf("%-25s %-25s %-10s\n", author, title, copies);
  printf("%-25s %-25s %-10s\n", array[0], array[1], array[2]);
  return printf("\n");
}

int main()
{
  char *line = newRecord("Pressman R", "Software Engineering", 5);
  printRecord(line);
  return 0;
}
```



18.3 Create a Text File

To create a file, we have to open it, write data to it, and then close it.

```
/* createFile: creates a text file */
int createFile()
{
  FILE *catalogue;
  char *line;

  catalogue = fopen("cat.data", "w");
  line = newRecord("Kernighan&Ritchie","C Programming Language",10);
  fprintf(catalogue, "%s\n", line);
  ...
  fclose(catalogue);
  return 0;
}
```

The line

```
FILE *catalogue;
```

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

We make the connection between the internal file name, *catalogue*, and the external file name, *cat.dat*, with

```
catalogue = fopen("cat.data", "w");
```

The external filename is the name of the file as it is known to the operating system, and is as seen in the folder or directory listing. *fopen()* opens the file and *"w"* (the mode) stands for writing data to the file. If the file already exists, it is overwritten with the new file. *fopen()* is defined in *stdio.h*.

We assemble a line with

```
line = newRecord("Kernighan & Ritchie","C Programming Language",10);
```

and write it to the file with

```
fprintf(catalogue, "%s\n", line);
```

fprintf() is defined in *stdio.h*. It works just like *printf()* does, except that data is written to a file rather than the screen. Note that we write a newline character after writing *line* to the file.

```
fclose(catalogue);
```

closes the file. *fclose()* flushes buffers (a buffer is a temporary storage area for data) and breaks the connection between the internal and external file names. *fclose()* is defined in *stdioh*.

Having created a text file you can look at it with a text editor such as Notepad (but not with a word processor such as Microsoft Word because it inserts formatting characters), and even make changes to it.

18.4 Print a Text File

To retrieve the contents of a file we have to perform a priming read, open the file, read data from it, and then close it when finished.

```
/* printfile: displays contents of a text file */
int printFile()
{
  FILE *catalogue;
  char *line;
  catalogue = fopen("cat.data", "r");
  line = fGetLine(catalogue, 80);
  while (!feof(catalogue)) {
    printRecord(line);
    line = fGetLine(catalogue, 80);
  }
  return fclose(catalogue);
}
```

The line

```
catalogue = fopen("cat.data", "r");
```

opens the file for reading. It creates a connection between the internal and external file names. The mode, *"r"*, stands for reading. You can retrieve records from a file that is opened for reading.

The first call to *fGetLine()*

```
line = fGetLine(catalogue, lineSize);
```

is the priming read. It attempts to read at most 80 characters of a line of text from the given file. If it fails to do so, it sets *feof()* to true.

feof() stands for end of file. We loop for as long as the end of the catalogue file has not been reached. We print the current line and attempt to retrieve the next line from the file.

fGetLine() requires the maximum length of the line it is to retrieve from the file. We have used a generous value, namely 80. In writing *printFile()* you need to be aware of what *createFile()* does.

What is this *fgetLine()*? Well, it is function similar to the *getString()* function written for input from the keyboard in the chapter on Functions. Only *fgetLine()* gets a line of text from a file that has already been opened.

```
/* fGetLine: reads a line of text from the given file */
char *fGetLine(FILE *file, int size)
{
  char *line = malloc(size + 1);
  int c, i;

  i = 0;
  while ((c = getc(file)) != EOF) {
    if (c == '\n')
      break;
    if (i < size - 1) {
      line[i] = c;
      i++;
    }
  }
  line[i] = '\0';
  return line;
}
```

fGetLine() receives the name of the file to read from, and the maximum length of the line to be retrieved. *getc()* gets a single character from the given file. If *getc()* fails to read a character, it sets the end of file indicator, *feof()* to true and returns *EOF*. *EOF*, a constant for End Of File defined in *stdio.h*, as is *getc()*.

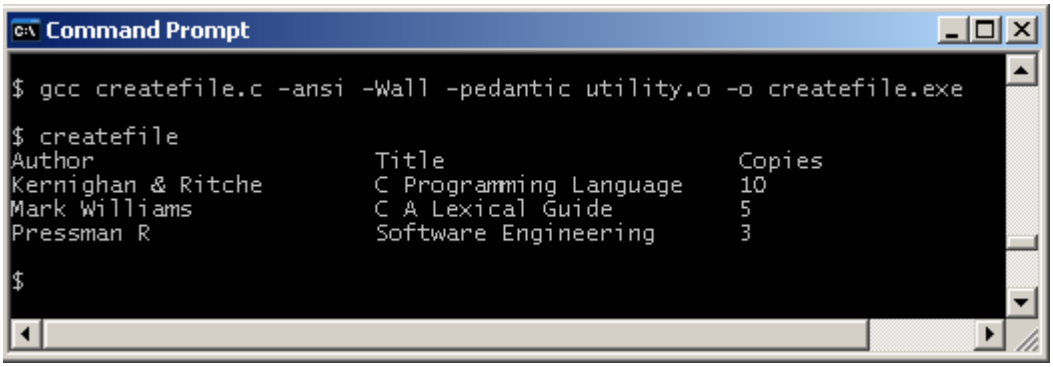
We loop for as long as the end of the file has not been reached. We retrieve the next character. If it is not the new-line character, and if we have not reached the end of the allocated storage for the line, we store the character in the line. If we have reached the end of the allocated storage, the character is ignored. If the character is a new-line character we have found the end of a line and break out of the loop. When the loop terminates we assign the null character to the line before returning it.

fgetLine() resides in *utlity.c*.

Here is the entire program and its run.

```
/* createfile.c: creates a text file, displays its contents */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include "utility.h"

/* newRecord: assembles a new line from
   the given author, title and number of copies */
char *newRecord(char *author, char *title, int nCopies)
{
  int size;
  char *line;
  char *copies = intToString(nCopies);

  size = strlen(author) + strlen(title) + strlen(copies);
  size = size + 3; /* +3 for , , and null */
  line = malloc(size);
  strcpy(line, author);
  strcat(line, ",");
  strcat(line, title);
  strcat(line, ",");
  strcat(line, copies);
  return line;
}

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

/* printRecord: splits line into components and prints them */
int printRecord(char *line)
{
  char *ptr;
  char* array[3];  /* indexed 0..2 */
  int i;

  for (ptr = strtok(line, ","), i = 0;
            ptr != NULL && i < 3;
            ptr = strtok(NULL, ","), i++)
    array[i] = ptr;
  printf("%-25s %-25s %-6s", array[0], array[1], array[2]);
  return printf("\n");
}
```

```
/* createFile: creates a text file */
int createFile()
{
  FILE *catalogue;
  char *line;

  catalogue = fopen("cat.data", "w");
  line = newRecord("Kernighan & Ritchie",
                               "C Programming Language", 10);
  fprintf(catalogue, "%s\n", line);
  line = newRecord("Mark Williams", "C A Lexical Guide", 5);
  fprintf(catalogue, "%s\n", line);
  line = newRecord("Pressman R", "Software Engineering", 3);
  fprintf(catalogue, "%s\n", line);
  return fclose(catalogue);
}

/* printfile: displays contents of a text file */
int printFile()
{
  FILE *catalogue;
  char *line;

  printHeadings();
  catalogue = fopen("cat.data", "r");
  line = fGetLine(catalogue, 78);
  while (!feof(catalogue)) {
    printRecord(line);
    line = fGetLine(catalogue, 78);
  }
  return fclose(catalogue);
}

int main()
{
  createFile();
  printFile();
  return 0;
}
```



18.5 Append to a Text File

We can add lines to the end of a text file.

First, we open the file in *append* mode.

```
catalogue = fopen("cat.data", "a");
```

Then we create a new record and write it to the file.

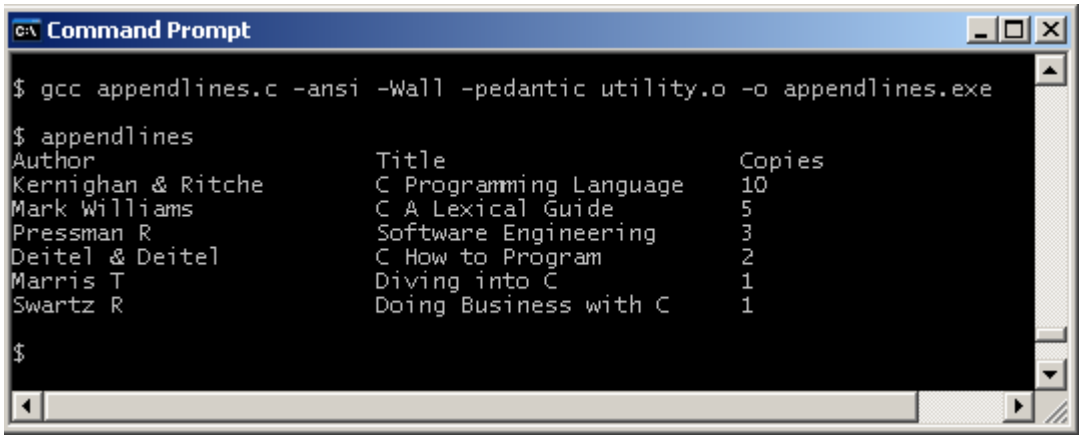
```
line = newRecord("Deitel & Deitel", "C How to Program", 2);
fprintf(catalogue, "%s\n", line);
```

Finally, we close the file.

Here is the entire program and its run.

```
/* appendlines.c: adds records onto the end of a textfile */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include "utility.h"

/* newRecord: assembles a new line from
   the given author, title and number of copies */
char *newRecord(char *author, char *title, int nCopies)
{
  int size;
  char *line;
  char *copies = intToString(nCopies);

  size = strlen(author) + strlen(title) + strlen(copies);
  size = size + 3; /* +3 for , , and null */
  line = malloc(size);
  strcpy(line, author);
  strcat(line, ",");
  strcat(line, title);
  strcat(line, ",");
  strcat(line, copies);
  return line;
}

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

```
/* printRecord: splits line into components and prints them */
int printRecord(char *line)
{
  char *ptr;
  char* array[3];  /* indexed 0..2 */
  int i;

  for (ptr = strtok(line, ","), i = 0;
            ptr != NULL && i < 3;
            ptr = strtok(NULL, ","), i++)
    array[i] = ptr;
  printf("%-25s %-25s %-6s", array[0], array[1], array[2]);
  return printf("\n");
}

/* addRecords: adds records to an existing file */
int addRecords(char *fileName)
{
  FILE *catalogue;
  char *line;

  catalogue = fopen(fileName, "a");
  line = newRecord("Deitel & Deitel", "C How to Program", 2);
  fprintf(catalogue, "%s\n", line);
  line = newRecord("Marris T", "Diving into C", 1);
  fprintf(catalogue, "%s\n", line);
  line = newRecord("Swartz R", "Doing Business with C", 1);
  fprintf(catalogue, "%s\n", line);
  fclose(catalogue);
  return 0;
}

/* printfile: displays contents of a text file. */
int printFile(char *fileName)
{
  FILE *catalogue;
  char *line;

  printHeadings();
  catalogue = fopen(fileName, "r");
  line = fGetLine(catalogue, 78);
  while (!feof(catalogue)) {
   printRecord(line);
   line = fGetLine(catalogue, 78);
  }
  return fclose(catalogue);
}

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



It would be an error to create a file with one external filename, and then to attempt to print a file with a different external filename. So, in *main()*, we supply the external filename to both *addRecords()* and *printFile()* as arguments - then you can see at a glance that you are referring to the same file on disk.

Notice that if you run this program several times, you get the same records added again and again to the file. You could run the original create file program to restore the situation.

18.6 Duplicate a Text File

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

```
/* duplicateFile: creates a copy of the original file and saves it
   as a new file */
int duplicateFile(char *fromFileName, char *toFileName)
{
  FILE *catalogue, *copy;
  char *line;

  catalogue = fopen(fromFileName, "r");
  copy = fopen(toFileName, "w");

  line = fGetLine(catalogue, 80);
  while (!feof(catalogue)) {
    fprintf(copy, "%s\n", line);
    line = fGetLine(catalogue, 80);
  }

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

We use this function in the next program.

18.7 Remove a Record From a Text 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;
  char *line;

  catalogue = fopen(fileName, "r");
  copy = fopen("temp.data", "w");

  line = fGetLine(catalogue, 80);
  while (!feof(catalogue)) {
   if (strstr(line, author) == NULL)
      fprintf(copy, "%s\n", line);
    line = fGetLine(catalogue, 80);
  }

  fclose(catalogue);
  fclose(copy);

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

  return 0;
}
```

*strstr(char *string1, char *string2)* looks for *string2* in *string1*. It returns *NULL* if *string2* is not in *string1*, a pointer to the beginning of *string2* in *string1* otherwise.

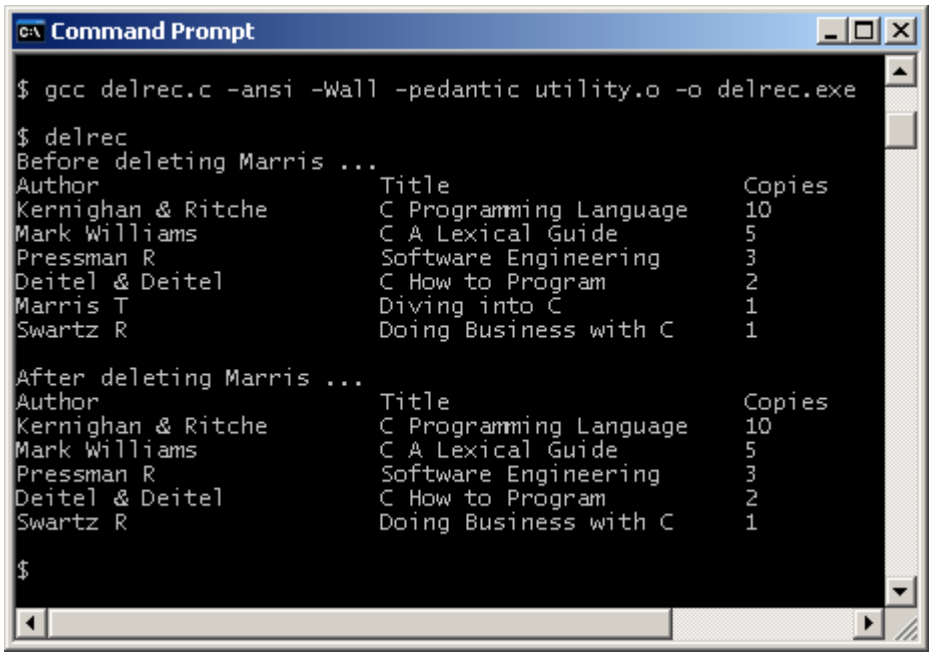
So we copy the original *cat.data* without the deleted record to *temp.data*. Then we copy the original *cat.data* to *old.data*, and the copy *temp.data* to *cat.data*.

Here is the entire program and its run.

```
/* delrec.c: deletes a record from a text file */

#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include "utility.h"

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

```
/* printRecord: splits line into components and prints them */
int printRecord(char *line)
{
  char *ptr;
  char* array[3];  /* indexed 0..2 */
  int i;

  for (ptr = strtok(line, ","), i = 0;
            ptr != NULL && i < 3;
            ptr = strtok(NULL, ","), i++)
    array[i] = ptr;
  printf("%-25s %-25s %-6s", array[0], array[1], array[2]);
  return printf("\n");
}

/* printfile: displays contents of a text file */
int printFile(char *fileName)
{
  FILE *catalogue;
  char *line;

  printHeadings();
  catalogue = fopen(fileName, "r");
  line = fGetLine(catalogue, 78);
  while (!feof(catalogue)) {
   printRecord(line);
   line = fGetLine(catalogue, 78);
  }
  return fclose(catalogue);
}

/* duplicateFile: creates a copy of the original file and saves it
   as a new file */
int duplicateFile(char *fromFileName, char *toFileName)
{
  FILE *catalogue, *copy;
  char *line;

  catalogue = fopen(fromFileName, "r");
  copy = fopen(toFileName, "w");

  line = fGetLine(catalogue, 80);
  while (!feof(catalogue)) {
    fprintf(copy, "%s\n", line);
    line = fGetLine(catalogue, 80);
  }

  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;
  char *line;

  catalogue = fopen(fileName, "r");
  copy = fopen("temp.data", "w");

  line = fGetLine(catalogue, 80);
  while (!feof(catalogue)) {
   if (strstr(line, author) == NULL)
      fprintf(copy, "%s\n", line);
    line = fGetLine(catalogue, 80);
  }

  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;
}
```



Notice that we printed the contents of the file before and again after the update. This is always an important thing to do because it shows at a glance that the update has taken place.

Exercise 18.1

1. A student has a unique five-digit reference number, a name, and a number of credits obtained by completing assignments. Create a text 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*.

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.

3. A student has passed another assignment and, so, their record needs updating. One way of achieving this is to delete the student's record and then append the amended record to the file. 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.

We have looked at text files. **Next** we look at binary files and what can go wrong with file processing.

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