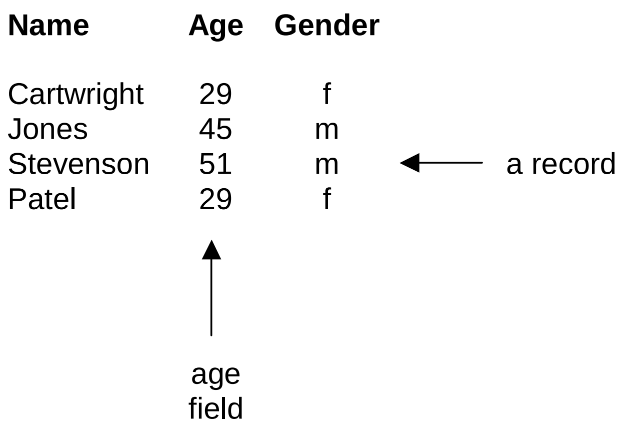
8 FILES

8.1 Files. Records and Fields

Every year The City holds a marathon. Runners apply to enter the competition by filling in a form and sending it, together with the fee, to The Organiser. The Organiser of The City Marathon maintains a computer file of all registered competitors. Part of the information recorded for each competitor is

name
age
gender

Part of the file might look like



Name	**Age**	**Gender**
Cartwright	29	f
Jones	45	m
Stevenson	51	m
Patel	29	f

Each attribute (name, age, gender) is an example of a field. The different fields together make up a single structure known as a record. In our example there is one record for each competitor. A file contains many records.

We can model the record in C as a fixed-length structure like this

```
typedef struct {
  char name[BUFSIZ];
  int age;
  char gender;
} CompetitorStructure;
```

The length or size of the structure is returned by the C *sizeof* function.

```
int structureSize =
               (int)sizeof(CompetitorStructure);
```

8.2 File Creation

To place records in a new file we

```
define file and record variables
open the file for writing
while not finished
  assemble a record from its field values
  write the record to the file
endwhile
close the file
```

So, to place competitor records (defined in section 8.1 above) on a new competitors file we write

```
int main()
{
  CompetitorStructure competitorRecord;
  FILE *competitorsFile = fopen("compet.dat", "wb");

  readCharArray("\nName (* to finish)? ",
                competitorRecord.name);
  while (competitorRecord.name[0] != '*') {
    competitorRecord.age = intNumberRead("Age? ");
    competitorRecord.gender = charRead("Gender (m/f)? ");

    fwrite(&competitorRecord,
                sizeof(CompetitorStructure),1,competitorsFile);

    readCharArray("\nName (* to finish)? ",
                competitorRecord.name);
  }
  fclose(competitorsFile);
  return 0;
}
```

CompetitorStructure is the record structure defined in section 8.1 above. The line

```
FILE *competitorsFile = fopen("compet.dat", "wb");
```

- defines a variable of type (pointer to) *FILE* named *competitorsFile*
- declares that the file is to be saved on disk under the name *compet.dat*
- opens a new file ready for writing binary data to it
- overwrites any existing file with the name *compet.dat*

fopen makes a connection between the variable file name used inside the program (*competitorsFile*) and the external file name used outside the program by the computer's operating system (*compet.dat*). The file open mode, *wb*, destroys any existing file with the name *compet.dat*.

The function call

```
readCharArray("\nName (* to finish)? ",
                    competitorRecord.name);
```

displays the prompt *Name (* to finish)?* on the screen and stores the user's response in the name field of *competitorRecord*.

We loop for as long as the first character stored in *competitorRecord.name* is not *. In the loop we fill the remaining fields with values, write the record to the file and ask for the next name to be input.

The statement

```
fwrite(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);
```

says write a copy of the data held in a competitors record into the competitors file. The arguments *sizeof(CompetitorStructure)* and *1* mean one item of size equal to the record structure.

Records are written onto the file sequentially, one record after another.

```
Cartwright 29 f  Jones     45 m Stevenson  51 m  Patel     29 f
```

Notice the repeating pattern in types of the data held:

array-of-char int char array-of-char int char array-of-char int char

(The fact that data is converted into binary format before being written to the file need not concern us here.) Finally, we close the file.

```
fclose(competitorsFile);
```

fclose breaks the connection between the variable file name *competitorsFile* and the external file name *compet.dat*; it also ensures that the last bit of data is physically written into the file.

FILE, *fopen*, *wb* and *fclose* are all defined in *stdio*.

8.3 Record Retrieval

To display the contents of a file we

```
define file and record variables
open the file for reading
retrieve the first record from the file
while the end of the file has not been reached
  display the contents of each field in the record
  retrieve the next record from the file
endwhile
close the file
```

To display the contents of the competitors file we write

```
void main(void)
{
  CompetitorStructure competitorRecord;
  FILE *competitorsFile = fopen("compet.dat", "rb");

  fread(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);
  while (!feof(competitorsFile)) {
    printf("%s  ", competitorRecord.name);
    printf("%d  ", competitorRecord.age);
    printf("%c  ", competitorRecord.gender);
    fread(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);

  }
  fclose(competitorsFile);
}
```

The line

```
FILE *competitorsFile = fopen("compet.dat", "rb");
```

- defines a variable of type (pointer to) *FILE* named *competitorsFile*
- declares that the file to be opened is named *compet.dat* on disk
- opens an existing file ready for retrieving binary data from it

The argument *rb* stands for read binary mode.

The statement

```
fread(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);
```

says read a record from the *competitorsFile* and place a copy of it in *competitorRecord*. The arguments *sizeof(CompetitorStructure)* and *1* mean one item of size equal to the record structure. Suppose the file was created thus

```
Cartwright 29 f  Jones     45 m Stevenson  51 m  Patel     29 f
```

then, when the first record is read, *Cartwright* would be placed in the *name* field, *29* in the *age* field and *f* in the *gender* field.

If the end of the file has been reached and there are no more records to be read, then *fread* fails to retrieve a record and *feof* (for end of file) is set to TRUE. So we write

```
while (!feof(competitorsFile)) {
   ...
}
```

8.4 Shared File and Record Declarations

A record retrieval program will fail if

- there is no file on disk to open and retrieve data from
- the record structure used in the file creation program is different from the record structure used in the record retrieval program

For example, if the file was created with the following record structure

array-of-char int char

then retrieving a record from the file and placing it in record variable of the form

int array-of-char char

is a recipe for creating garbage. (You try fitting an array of characters into a single *int* variable.) Therefore it would be sensible if both file creation and record retrieval programs shared the same copy of the external file name and record structure. So we create a copy of these declarations, using our usual program editor, and save it with the name *compet.h* (say). This is what *compet.h* looks like.

```
/* compet.h - contains competitor file record structure */
              and external file name. */

#include <stdio.h>
#define competitorDiskFileName "compet.dat"

typedef struct {
  char name[BUFSIZ];
  int age;
  char gender;
} CompetitorStructure;
```

We do not compile or run *compet.h*, but we *#include* it in our programs as shown below.

```
/* program 8.1 - sets up a Marathon File with competitors. */

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

char charRead(char prompt[]);
/* post-condition: returns a char entered at the keyboard. */

int intNumberRead(char prompt[]);
/* post-condition: returns a number entered at the keyboard.
*/

void readCharArray(char prompt[], char array[]);
/* post-condition: array' contains a string entered at the
          keyboard. */
```

```
int main()
{
  CompetitorStructure competitorRecord;
  FILE *competitorsFile = fopen(competitorDiskFileName, "wb");

  printf("File creation program.\n");
  readCharArray("\nName (* to finish)? ",
                competitorRecord.name);
  while (competitorRecord.name[0] != '*') {
    competitorRecord.age = intNumberRead("Age? ");
    competitorRecord.gender = charRead("Gender (m/f)? ");
    fwrite(&competitorRecord,
                sizeof(CompetitorStructure),1,competitorsFile);
    readCharArray("\nName (* to finish)? ",
                competitorRecord.name);
  }
  fclose(competitorsFile);
  printf("\nFinished.\n");
  return 0;
}

char charRead(char prompt[])
{
  char charArray[BUFSIZ];
  printf("%s", prompt);
  gets(charArray);
  return charArray[0];
}

int intNumberRead(char prompt[])
{
  char charArray[BUFSIZ];
  int number = 0;
  printf("%s", prompt);
  gets(charArray);
  sscanf(charArray, "%d", &number);
  return number;
}

void readCharArray(char prompt[], char charArray[])
{
  printf("%s", prompt);
  gets(charArray);
}
```

An example of a program run is

```
File creation program.

Name (* to finish)? Cartwright
Age? 29
Gender (m/f)? f

Name (* to finish)? Jones
Age? 45
Gender (m/f)? m

Name (* to finish)? *

Finished.
```

At compile time, the line *#include "compet.h"* is replaced with the contents of *compet.h*. Notice the use of quotation marks rather than angle brackets to delimit the name of the text for inclusion; usually, the quotation marks mean that the text to be included is found in the same place as the program source text. When the contents of an included text file is changed, then all the programs that *#include* the text file must be re-compiled.

Any named collection of data which is stored on disk is known as a file. So program source text and headers are referred to as files. An executable program held on disk is also known as a file. And a file is also a collection of records.

Now we address the problem of attempting to read from a file which does not exist. Fortunately, if a file cannot be opened, *fopen* returns *NULL*. So we can write

```
FILE *competitorsFile = fopen(competitorDiskFileName, "rb");
if (competitorsFile == NULL) {
  printf("\nProgram halted: unable to open %s\n",
               competitorDiskFileName);
  exit(EXIT_FAILURE);
}
```

exit is a C keyword. It halts program execution, flushing buffers and closing files as it does so. Both *exit* and *EXIT_FAILURE* are defined in *stdlib*. *exit* and *EXIT_FAILURE* are used in program 8.2 shown below.

```
/* program 8.2 - Displays contents of Marathon Entries */
/*               File.                                 */

#include <stdio.h>
#include <stdlib.h>
#include "compet.h"

int main()
{
  FILE *competitorsFile;
  CompetitorStructure competitorRecord;

  competitorsFile = fopen(competitorDiskFileName, "rb");
  if (competitorsFile == NULL) {
    printf("Program halted: unable to open %s.\n",
               competitorDiskFileName);
    exit(EXIT_FAILURE);
  }
  fread(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);
  while (!feof(competitorsFile)) {
    printf("%s  ", competitorRecord.name);
    printf("%d  ", competitorRecord.age);
    printf("%c  ", competitorRecord.gender);
    printf("\n");
    fread(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);
  }
  fclose(competitorsFile);
  return 0;
}
```

A result of running program this program us

```
Cartwright  29  f
Jones  45  m
Stevenson  51  m
Patel  29  f
```

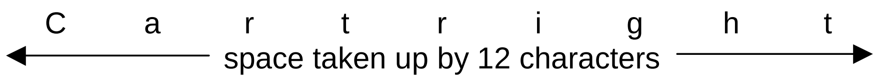
is shown on the screen. The appearance would be improved if we could arrange the output in columns thus

```
    Cartwright      29      f
    Jones           45      m
    Stevenson       51      m
    Patel           29      f
```

This can be done if we specify a space on the screen in which the contents of a field is to be displayed. For example, the statement

```
printf("%-12s  ", competitorRecord.name);
```

says print the contents of *competitorRecord.name* right justified in a space twelve characters wide. For example



Similarly,

```
printf("%4d", competitorRecord.age);
```

says print the contents of *competitorRecord.Age* right justified in a space four characters wide.

And

```
printf("%3c", competitorRecord.gender);
```

says print the contents of *competitorRecord.gender* right justified in a space 3 characters wide.

The space allocated for printing a value is known as the field width. A signed integer preceding the conversion specification character is known as the field width specifier. A minus sign indicates left justified; no sign or a plus sign indicates right justified. So, for example

"%-10s" means display a string value left justified in a field width of 10
"%10s" means display a string value right justified in a filed width of 10.

8.5 Data Dictionaries

We specify the file content and record structure in a Data Dictionary. For example

DATA DICTIONARY

Author: Terry Marris **Project:** **Date:** 8 May 1994

Data Store Name: compet.dat

Description: file of competitor records

Data Name	**Type**	**Comments**
competitorFile	FILE	File of competitor records
competitor	CompetitorStructure	one per competitor
name	array of char	
age	int	
gender	char	values 'm' or 'f'

Figure 8.1 - Data Dictionary specifying the competitor file structure and content.

The data store name is the name of the file on disk. The description is a general description of the file, its content and purpose. Under Data Name we list the file, structure and member variable names, and, under the Type heading, their types. The comments column includes explanations of the various entries.

Exercise 8.1

Remember to use data dictionaries to specify your recorda and files.

1 The Local Constabulary wishes to create a register of bicycles and their owners as part of their efforts to detect and deter theft and to return recovered cycles. The data to be recorded includes owner's name, date of birth, post code, house number, bicycle frame number and bicycle frame type (for example, diamond, open or tandem). Write and test two programs, one to create the register (that is, file) with about seven records, and one to display the contents of the file neatly in columns; each column should have its own heading.

2 A Network Manager maintains a central file of the software packages used within a college. The data recorded for each package includes its name, type (word processor, spreadsheet, database or compiler for example), licence number, number of licensed copies and locations installed. Write and test two programs, one to create the file and one to display its contents in tabular format (that is, in a table).

8.6 File to Printer

We need to print out the contents of the Marathon Entries File; copies are needed for The Organiser (so that all the entry details may be checked), for the finish marshals (so that they can produce the results of the race) and for the commentator (so that runners of note may be identified). Program 8.3 displays the contents of the Marathon Entries File on the printer.

```
/* program 8.3 - Displays contents of Marathon Entries File */
/*               on the printer.                            */

#include <stdio.h>
#include <stdlib.h>
#include "compet.h"
#define printerDevice "LPT1"

int main()
{
  FILE *printer = fopen(printerDevice, "w");
  FILE *competitorsFile = fopen(competitorDiskFileName, "rb");
  CompetitorStructure competitorRecord;

  if (printer == NULL) {
    printf("Program halted: unable to access printer\n.");
    exit(EXIT_FAILURE);
  }
  if (competitorsFile == NULL) {
    printf("Program halted: unable to open %s.\n",
               competitorDiskFileName);
    exit(EXIT_FAILURE);
  }
  fread(&competitorRecord,sizeof(CompetitorStructure),1,
               competitorsFile);
  while (!feof(competitorsFile)) {
    fprintf(printer, "%-12s", competitorRecord.name);
    fprintf(printer, "%4d", competitorRecord.age);
    fprintf(printer, "%3c", competitorRecord.gender);
    fprintf(printer, "\n");
    fread(&competitorRecord, sizeof(CompetitorStructure), 1,
               competitorsFile);
   }
   fclose(competitorsFile);
   fflush(printer);
   fclose(printer);
   return 0;
}
```

First, we give a name to the printer device. On many small computer systems this device is known as LPT1 (1 - the number not the letter); it may be different on your computer system.

```
#define printerDevice "LPT1"
```

The printer is regarded as a file. And so it is defined, opened and closed just like a file is.

```
FILE *printer = fopen(printerDevice, "w");
…
fclose(printer);
```

Notice that the open mode is *"w"*; this stands for write text mode. The statement

```
fflush(printer);
```

ensures that the last record is completely written out onto the printer and that the printer is set up ready for its next input.

fprintf converts and formats its arguments and outputs text just like *printf* does. But *fprintf* outputs its text onto the named file. In this example, the named file represents the printer.

```
fprintf(printer, "%-12s  ", competitorRecord.name);
```

Exercise 8.2

1 Write and test a program which will display the contents of the bicycle file, created in exercise 8.1 above, on the printer. Note: if your printer is attached to a network, you might (or you might not) need to press the three keys marked Ctrl Alt and PrintScreen simultaneously to force the output to the printer; this should be done when your program run has finished.

2 When the police recover a bicycle, they might wish to contact the owner. Write and test a program which will input a bicycle frame number, search through the bicycle file created in program 8.1 above, and, if found in the file, outputs the owner's name and date of birth. Your program should output a suitable message if a record for the recovered bicycle is not found in the file. A possible method is

```
input theFrameNumber
store FALSE in found
open the bicycleFile
retrieve the first bicycleRecord
while not at the end of the bicycleFile
  if theFrameNumber is the same as the bicycleRecord.frameNumber then
    store TRUE in found
    display the bicycleRecord
  endif
  retrieve the next bicycleRecord
endwhile
close the bicycleFile
if found still contains the value FALSE
   display "No registered owner found for this bicycle."
endif
```

8.7 Maintaining a File

The contents of files do not usually remain the same for long. For example, in a file of entries for The Marathon, some entries will be withdrawn, some new ones included, some people may change their name or address or corrections to some records might be needed. Let us list some of the operations The Organiser might wish to perform on the Competitors' File.

- remove a competitor's record
- include a new competitor's record
- print out all the competitor records in record number order
- print out the competitors records for a given category
- find a competitor's record from their competitor number
- change a competitor's record

The Organiser's view of the program is a menu which provides the required operations. We shall consider the programmers view of the program shortly. But for now we shall consider just the user's view.

Let us design, write and test a general-purpose function which displays a menu and returns a valid choice selected by the user at the keyboard.

```
char choiceFromMenu(char menu[], char validChoices[]);
/* Displays menu on the screen.
   post-condition: returns a value from validChoices
                   selected by the user at the keyboard. */
```

An example of a call to this function is

```
char menu[] = "             Menu"
              "     1  Add a new competitor\n"
              "     2  Remove a competitor\n"
              "     3  List all competitors\n"
              "     9  Quit\n\n"
              "     Your choice? ";

char validChoices[] = "1,2,3,9";
char choice = choiceFromMenu(menu, validChoices);
```

The C compiler joins together (that is, concatenates) adjacent string literals. So

```
              "             Menu"
              "     1  Add a new competitor\n"
              "     2  Remove a competitor\n"
              "     3  List all competitors\n"
              "     9  Quit\n\n"
              "     Your choice? ";
```

becomes one long string.

Now let us look at the function itself. Obtaining a choice from the keyboard is easy. The problem is determining whether the value input is also in the string of valid choices. The *strchr* function does just that. An example of a call to strchr is

```
returnedValue = strchr(string, character);
```

The *strchr* function from the string.h library searches for character (including the NULL character) within *string*. It returns NULL if character is not found in *string*, otherwise, it returns the address of the first occurrence of character in *string*. *strchr* is used in the *choiceFromMenu* function shown below.

```
char choiceFromMenu(char menu[], char validChoices[])
{
  enum { false };

  char choice[BUFSIZ];
  int choiceIsOK = false;

  while (!choiceIsOK) {
    printf("%s", menu);
    gets(choice);
    choiceIsOK = ((strlen(choice) != 0) &&
                  (strchr(validChoices, choice[0]) != NULL));
  }
  printf("\n");
  return choice[0];
}
```

Now, why check for string length equal to zero? If the user just pressed return in response to the menu of options, then choice would contain just the end-of-string character - NULL. *validChoices* also contains the end-of-string character. We do not want this character to be regarded as a valid choice. So we ensure that if the length of the string is zero (remember that the NULL character is not counted) then the user's choice is not valid.

Based on The Organiser's view we might draw the program structure chart like this:

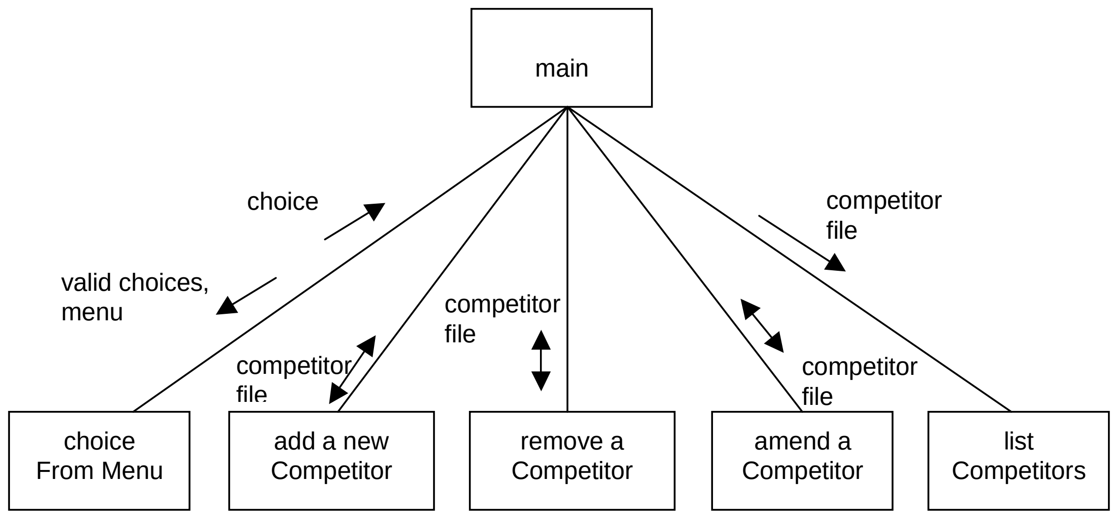


Figure 8.2 - possible program structure based on the organiser's view

Let us look at the problem from the programmer's point of view. We are basically dealing with competitor structure variables and the competitors file. What do we need to do?

- fill a competitor structure variable with values from the keyboard
- write the contents of a competitor structure variable on the screen
- fill a competitor structure variable with values from the competitor file
- write the contents of a competitor structure variable onto the competitor file

Let us design and write functions to do these tasks. We start with *competitorFromKeyboard*. Its specification is

```
CompetitorStructure competitorFromKeyboard(void);
/* Post-condition: returns a competitor record entered
                   at the keyboard.                    */
```

An example of a call to *competitorFromKeyboard* is

```
CompetitorStructure newCompetitor = competitorFromKeyboard();
```

Its implementation is very straightforward.

```
CompetitorStructure competitorFromKeyboard()
{
  CompetitorStructure competitor;

  readArrayOfChar("Competitor name? ", competitor.name);
  competitor.age = intNumberRead("Age? ");
  competitor.gender = charRead("Gender (m/f)? ");
  return competitor;
}
```

To write a competitor's record on the screen is also straightforward and is not described here.

Now we look at retrieving a competitor from the file. The Organiser identifies each competitor by their competitor number; no two competitors have the same number. Fortunately for us as programmers, C automatically numbers each structure (or record) in a file, starting with zero.

record Number	**name**	**age**	**gender**
0	Cartwright	29	f
1	Jones	45	m
2	Stevenson	51	m
3	Patel	29	f

We make the convenient connection between competitor number and record number: they both represent the same thing. We use the record number to directly access a particular record without first retrieving and examining every preceding record. The *stdio* library function *fseek* does this for us.

fseek sets the file-position indicator: this is the point in the file where the next read or write is to take place. An example of a call to *fseek* is

```
fseek(competitorFile,
     recordNumber * sizeof(CompetitorStructure), SEEK_SET);
```

The effect of *SEEK_SET* is to set the beginning of the file as the starting point for the file-position indicator. *SEEK_SET* is defined in the *stdio* library.

We use *fseek* in the implementation of *competitorFromFile*.

```
CompetitorStructure competitorFromFile(FILE *competitorFile,
                                       long recordNumber)
{
  CompetitorStructure competitor;
  fseek(competitorFile,
     recordNumber * sizeof(CompetitorStructure),SEEK_SET);
  fread(&competitor, sizeof(CompetitorStructure), 1,
     competitorFile);
  return competitor;
}
```

long stands for *long int*, a possibly large integer.

There is one important restriction: it would be an error to seek past the end of the file. We have to decide upon, and fix, the maximum size of the file and hence the last record number before we use *fseek*. So, we modify *compet.h* accordingly and create a file of blank records (otherwise, how can the last record number exist?).

```
/* compet.h - contains competitor file record structure
              and external file name.                   */
#include <stdio.h>
#define competitorDiskFileName "compet.dat"

typedef struct {
  char name[BUFSIZ];
  int age;
  char gender;
} CompetitorStructure;

#define lastRecordNumber 10L
```

Here, for our convenience, we have defined the last record number to be ten. In practice, of course, the last record number would be several hundred or even thousand.

```
/* program 8.4 - creates a file of blank competitor records.
*/
#include <stdio.h>
#include "compet.h"

int main()
{
  CompetitorStructure competitor = { "x", 0, 'x' };
  FILE *competitorFile = fopen(competitorDiskFileName, "wb");
  long recordNumber = 0L; /* long is short for long int */

  while (recordNumber <= lastRecordNumber) {
    fwrite(&competitor, sizeof(CompetitorStructure), 1,
          competitorFile);
    recordNumber++;
  }
  fclose(competitorFile);
}
```

Look at the line

```
CompetitorStructure competitor = { "x", 0, 'x' };
```

Here, the name field is initialised with the string *x*, (notice the double quotes) the age field with zero and the gender field with the character *x* (notice the single quotes).

Now we write the specification for *competitorFromFile*.

```
CompetitorStructure competitorFromFile(
          const FILE *competitorFile, long recordNumber);
/* Pre-condition: recordNumber is between 0 and
                  lastRecordNumber inclusive.
   Post-condition: returns the record at location recordNumber
                  in competitor file. */
```

The converse of *competitorFromFile* is *writeCompetitorToFile*. The specification for *writeCompetitorToFile* is

```
void writeCompetitorToFile(FILE *competitorFile,
                           CompetitorStructure competitor,
                           long recordNumber);
/* Post-condition: competitorFile' record located at
                  recordNumber is replaced with competitor. */
```

And here is its implementation

```
void writeCompetitorToFile(FILE *competitorFile,
                           CompetitorStructure competitor,
                           long recordNumber)
{
  fseek(competitorFile,
     recordNumber * sizeof(CompetitorStructure), SEEK_SET);
  fwrite(&competitor, sizeof(CompetitorStructure), 1,
          competitorFile);
}
```

Now for the pay off. We shall amend a competitor's record by first retrieving it from the file and then replacing it with the updated record. The record to be replaced is identified by the record or competitor number.

```
/* Change a competitor's record. */
recordNumber = recordNumberFromKeyboard();
competitor = competitorFromFile(competitorFile, recordNumber);
writeCompetitorToScreen(competitor);
competitor = competitorFromKeyboard();
writeCompetitorToFile(competitorFile, competitor,
                    recordNumber);
```

The implementation of other organiser-defined requirements are just as easy. To remove a competitor's record, we overwrite it with the null record, as used in the file creation program.

```
/* Remove a competitor. */
CompetitorStructure nullRecord = { "x", 0, 'x' };
recordNumber = recordNumberFromKeyboard();
writeCompetitorToFile(competitorFile, nullRecord,
                    recordNumber);
```

We include a new competitor in the file by finding the first record number whose name field contains the string "x" and then overwrite the record in the file with the new competitor details.

```
/* Add a new competitor. */
recordNumber = nextFreeRecordNumber(competitorFile);
if (recordNumber < 0L)
  printf("No room in file for another competitor\n");
else {
  newCompetitor = competitorFromKeyboard();
  writeCompetitorToFile(competitorFile, newCompetitor,
                    recordNumber);
}
```

To list all competitors, we retrieve the data stored for each record number in turn.

```
/* List all competitors. */
recordNumber = 0L;
while (recordNumber <= lastRecordNumber) {
  competitor = competitorFromFile(competitorFile,
                       recordNumber);
  printf("%3ld  ", recordNumber);
  writeCompetitorToScreen(competitor);
  recordNumber++;
}
```

However, to find a competitor's number when given just their name, we have to search through the entire file from beginning to end, record-by-record. Every time we find a matching name we print the record together with its corresponding record number. We set the file-position indicator to the beginning of the file with rewind

```
rewind(competitorFile);
```

And then process the entire file using the ordinary sequential method described in section 8.2 above.

```
/* Find a competitor's number. */
recordNumber = 0L;
readArrayOfChar("Competitor's name? ", aName);
rewind(competitorFile);
fread(&competitor, sizeof(CompetitorStructure), 1,
competitorFile);
while (!feof(competitorFile)) {
  if (strcmp(competitor.name, aName) == 0) {
    printf("%3ld  ", recordNumber);
    writeCompetitorToScreen(competitor);
  }
  recordNumber++;
  fread(&competitor, sizeof(CompetitorStructure), 1,
                  competitorFile);
}
```

Here is the entire program.

```
/* program 8.5 - maintains competitor file. */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "compet.h"

char charRead(char prompt[]);
/* Post-condition: returns a char entered at the keyboard. */

char choiceFromMenu(char menu[], char validChoices[]);
/* Displays menu, returns a valid choice. */
```

```
CompetitorStructure competitorFromFile(FILE *competitorFile,
                                       long recordNumber);
/* Pre-condition: recordNumber is between 0 and
                  lastRecordNumber inclusive.
   Post-condition: returns the record at location recordNumber
                  in competitor file. */

CompetitorStructure competitorFromKeyboard(void);
/* Post-condition: returns a competitor record entered
                   at the keyboard. */

void getCategory(int *min, int *max);
/* *min' and *max' contain values input by the user at the
keyboard. */

int  intNumberRead(char prompt[]);
/* Post-condition: returns number entered at the keyboard. */

long longNumberRead(char prompt[]);
/* Post-condition: returns number entered at the keyboard. */


long nextFreeRecordNumber(FILE *competitorFile);
/* Post-condition: returns the next free competitor number in
                   competitorFile - if there is one, otherwise
                   returns -1. */

void readArrayOfChar(char prompt[], char array[]);
/* Post-condition: array' contains characters entered at the
                   keyboard. */

long recordNumberFromKeyboard(void);
/* Post-condition: returns a number between 0 and
                   lastRecordNumber inclusive input by the
                   user at the keyboard. */

void writeCompetitorToFile(FILE *competitorFile,
                           CompetitorStructure competitor,
                           long recordNumber);
/* Post-condition: competitorFile' record located at
                   recordNumber is replaced with competitor.*/

void writeCompetitorToScreen(CompetitorStructure competitor);
/* Displays competitor on screen. */

int main()
{
  enum { false, true };

  char aName[BUFSIZ];
  int minAge, maxAge;
  long recordNumber;

  FILE *competitorFile = fopen(competitorDiskFileName, "r+b");
  CompetitorStructure competitor;
  CompetitorStructure nullRecord = { "x", 0, 'x' };
```

```
char menu[] = "\n\n"
"                    Menu\n\n"
"          1  Add a new competitor\n"
"          2  Remove a competitor\n"
"          3  List all competitors in competitor number
                  order\n"
"          4  List all competitors in a category\n"
"          5  Find a competitor's number\n"
"          6  Change a competitor's details\n"
"          9  Quit\n\n"
"          Your choice? ";

char validChoices[] = "1,2,3,4,5,6,9";
char choice = choiceFromMenu(menu, validChoices);

int isDone = (choice == '9');


if (competitorFile == NULL) {
  printf("Program halted: unable to open %s\n",
             competitorDiskFileName);
  exit(EXIT_FAILURE);
}
while (!isDone) {
  switch (choice) {

  case '1':  /* Add a new competitor. */
    recordNumber = nextFreeRecordNumber(competitorFile);
    if (recordNumber < 0L)
      printf("No room in file for another competitor\n");
    else {
      competitor = competitorFromKeyboard();
      writeCompetitorToFile(competitorFile, competitor,
                  recordNumber);
    }
    break;

  case '2':  /* Remove a competitor. */
    recordNumber = recordNumberFromKeyboard();
    writeCompetitorToFile(competitorFile, nullRecord,
                  recordNumber);
    break;

  case '3':  /* List all competitors. */
    recordNumber = 0L;
    while (recordNumber <= lastRecordNumber) {
      competitor = competitorFromFile(competitorFile,
                  recordNumber);
      printf("%3ld  ", recordNumber);
      writeCompetitorToScreen(competitor);
      recordNumber++;
    }
    break;
    case '4':  /* List all competitors in a category. */
      getCategory(&minAge, &maxAge);
      recordNumber = 0L;
      while (recordNumber <= lastRecordNumber) {
        competitor = competitorFromFile(competitorFile,
                     recordNumber);
        if  ((competitor.age >= minAge) && (competitor.age <=
                     maxAge)) {
          printf("%3ld  ", recordNumber);
          writeCompetitorToScreen(competitor);
        }
        recordNumber++;
      }
      break;

    case '5':  /* Find a competitor's number. */
      recordNumber = 0L;
      readArrayOfChar("Competitor's name? ", aName);
      rewind(competitorFile);
      fread(&competitor, sizeof(CompetitorStructure), 1,
                     competitorFile);
      while (!feof(competitorFile)) {
        if (strcmp(competitor.name, aName) == 0) {
          printf("%3ld  ", recordNumber);
          writeCompetitorToScreen(competitor);
        }
        recordNumber++;
        fread(&competitor, sizeof(CompetitorStructure), 1,
                     competitorFile);
      }
      break;

    case '6':  /* change a competitor's details. */
      recordNumber = recordNumberFromKeyboard();
      competitor = competitorFromFile(competitorFile,
                     recordNumber);
      writeCompetitorToScreen(competitor);
      competitor = competitorFromKeyboard();
      writeCompetitorToFile(competitorFile, competitor,
                     recordNumber);
    break;

    case '9':  /* Quit. */
      isDone = true;
      break;

    default:
      printf("main: This should not happen!\n");
      break;
    }
    choice = choiceFromMenu(menu, validChoices);
    isDone = (choice == '9');
  }
   return 0;
}
```

```
char charRead(char prompt[])
{
  char string[BUFSIZ];
  printf("%s", prompt);
  gets(string);
  return string[0];
}

char choiceFromMenu(char menu[], char validChoices[])
{
  enum { false };
  char choice[BUFSIZ];
  int choiceIsOK = false;

  while (!choiceIsOK) {
    printf("%s", menu);
    gets(choice);
    choiceIsOK = ((strlen(choice) != 0) &&
                  (strchr(validChoices, choice[0]) != NULL));
  }
  printf("\n");
  return choice[0];
}

CompetitorStructure competitorFromFile(FILE *competitorFile,
                                       long recordNumber)
{
  CompetitorStructure competitor;
  fseek(competitorFile,
     recordNumber * sizeof(CompetitorStructure),SEEK_SET);
  fread(&competitor, sizeof(CompetitorStructure), 1,
               competitorFile);
  return competitor;
}

CompetitorStructure competitorFromKeyboard(void)
{
  CompetitorStructure competitor;
  readArrayOfChar("Competitor name? ", competitor.name);
  competitor.age = intNumberRead("Age? ");
  competitor.gender = charRead("Gender (m/f)? ");
  return competitor;
}

void getCategory(int *min, int *max)
{
  char category;
  category = tolower(charRead(
           "Category - junior, regular or senior (j/r/s)? "));
  if (category == 'j')
    *min = 10, *max = 15;
  else if (category == 'r')
    *min = 16, *max = 39;
  else *min = 40, *max = 99;
  printf("\n");
}
```

```
int  intNumberRead(char prompt[])
{
  char string[BUFSIZ];
  int number = 0;
  printf("%s", prompt);
  gets(string);
  sscanf(string, "%d", &number);
  return number;
}

long longNumberRead(char prompt[])
{
  char string[BUFSIZ];
  long number = 0L;
  printf("%s", prompt);
  gets(string);
  sscanf(string, "%ld", &number);
  return number;
}

long nextFreeRecordNumber(FILE *competitorFile)
{
  CompetitorStructure competitor;
  long recordNumber = 0L;

  rewind(competitorFile);
  fread(&competitor, sizeof(CompetitorStructure), 1,
             competitorFile);
  while (!feof(competitorFile)) {
    if (strcmp(competitor.name, "x") == 0)
      return recordNumber;
    recordNumber++;
    fread(&competitor, sizeof(CompetitorStructure), 1,
             competitorFile);
  }
  return -1;  /* no free record numbers left in the file. */
}

void readArrayOfChar(char prompt[], char array[])
{
  printf("%s", prompt);
  gets(array);
}

long recordNumberFromKeyboard()
{
  long n = longNumberRead("Competitor number? ");
  while ((n < 0) || (n > lastRecordNumber)) {
    printf("Competitor number must be between 0 and %ld\n",
            lastRecordNumber);
    n = longNumberRead("Competitor number? ");
  }
  return n;
}
```

```
void writeCompetitorToFile(FILE *competitorFile,
                           CompetitorStructure competitor,
                           long recordNumber)
{
  fseek(competitorFile,
     recordNumber * sizeof(CompetitorStructure),SEEK_SET);
  fwrite(&competitor, sizeof(CompetitorStructure), 1,
               competitorFile);
}

void writeCompetitorToScreen(CompetitorStructure competitor)
{
  printf("%s,  %d,  %c\n", competitor.name, competitor.age,
                           competitor.gender);
}
/****** End Of Program 8.4 ******/
```

In main, the file is opened with

```
FILE *competitorFile = fopen(competitorDiskFileName, "r+b");
```

"r+b" specifies that a file, which already exists, is to be updated. So, before running program 8.5, we must first create the competitorFile by running program 8.4.

In the *getCategory* function, we have used the comma operator to help make the layout clearer. (Without the comma operator, braces would be needed to group each pair of assignment statements.)

```
if (category == 'j')
  *min = 10, *max = 15;
else if (category == 'r')
  *min = 16, *max = 39;
else
  *min = 40, *max = 99;
```

The success of the *nextFreeRecordNumber* function depends on the name field containing just *"x"*. So this function is tightly coupled to the file creation program and to the processes (in *main*) which remove a competitor's record from the file.

The validation in the *recordNumberFromKeyboard* function is essential because it is up to us to ensure that a record actually exists for the record number specified.

8.8 Documentation

File content and record structure are described in data dictionaries - see Figure 8.1 for an example.

8.9 Programming Principles

Perhaps the best way of managing files is to

- fix their maximum size at the beginning
- create the file with dummy records
- put data onto the file by overwriting a dummy record
- remove data from the file by overwriting the record with dummy data
- access the records either sequentially or directly as required

Remember that direct access methods can be used only on files which already exist.

When you need to perform the same operation on each and every record in a file, you will need to use a sequential access method: start with the first record in the file, retrieve and process it and then go on to the next record. Repeat the process for each record in turn.

When you need to perform an operation on a particular record in the file, you will need to use a direct access method: use *fseek* to set the file position indicator to the required record number. Remember that both *fread* and *fwrite* automatically advance the file position indicator after they have retrieved or written a record.

When trying to decide what functions are needed, concentrate on the data involved. For each data type, consider providing functions which input values from the keyboard and output them on the screen. Provide functions which deal only with records or structures. And supply functions which control file input and output.

Develop your programs incrementally, that is, repeatedly add a little bit to your program and then test it. Start with the user's view of the program, for example, the menu. Then for each function provide a temporary implementation - just a message announcing the name of the function and inviting the user to press a key to return to the menu. In this way you can test the menu before proceeding any further. Then take a function - any will do - complete and test it. Repeat for each function. This means that you are never very far away from a program which compiles and runs. It may not do all that it is supposed to do. But at least it will run and you will know which parts of it require completion.

Exercise 8.3

1 The Skip Hire Company (SHC) rent out skips for hire by trade and public alike. When a person wishes to hire a skip, they contact the SHC Office and give their name, address, nature of waste to be disposed (eg garden waste, builder's rubble) and the size of the skip required (eg mini, standard or maxi). SHC allocate a skip to the customer and deliver it to them on the required day. When the customer has filled the skip, they contact SHC who then collect it and dispose of its contents at an appropriate site. The skip is then allocated to the pool of available skips. If a skip becomes irreparably damaged then it is scrapped and replaced with a new skip. Every skip is replaced after being used for three years.

(a) Design a record structure for storing data on any one skip.

(b) Design and implement a menu that will enable the Skip Hire Manager to manage the allocation, delivery, collection and replacement of skips. The manager should also be able to print out the location of each skip.

(c) Create a file of about twenty blank or dummy skip records designed in part (a) above

(d) Complete a program which will maintain a file of skip records. The program should include functions which will add a new skip to the file, remove an old skip from the file and amend the contents of a skip record.

2 Create a file of about one hundred employee records. Each employee record should contain name, department and annual salary details. Then design and write a program which will print the contents of the file on the printer. The printer output should be paged. There should be a heading on each page, as well as a page number and column headings. At the bottom of each page there should be a summary total of the salaries on that page. In addition, the last page should include the total number of employees and the total annual salary bill. Of course, the headings should not creep up (or down) successive pages. You will need to decide on the maximum number of records to be printed on a page. One way of proceeding might be

```
retrieve first record from file
while not at end of file
    if (at top of page) or (past last record position on page) then
        new page functions
    endif
    print a record
    if (past last record position on page) then
        bottom of page functions
    endif
    retrieve next record from file
endwhile
```