Programming Project

Terry Marris January 2010

5 Entry

An entry has a name, an address, a category: either half marathon or full marathon, and a status: either current or withdrawn. We consider just a single entry before going on to the collection of entries.

5.1 entry.h

In *entry.h* we specify the entry structure and the constraints on its members, along with functions that operate on the members. It is always useful to include NOTE and TO DO notes as they occur to you during development. And it is also useful to identify listings with a version number and a date the file was last updated. You might like to include a history of amendments to impress an assessor or inform a colleague.

```
/* entry.h: marathon entry interface */
/* ver 1.5 11Jan2011 */

#ifndef ENTRY_H
#define ENTRY_H

#define MINENTRYNUMBER 1
#define MAXENTRYNUMBER 9999

#define MAXNUMBERLENGTH 4
#define NUMBERSIZE 6
#define NAMESIZE 20
#define ADDRESSSIZE 40

typedef enum { full, half, unknownCategory } Category;
typedef enum { current, withdrawn, unknownStatus } Status;

/* NOTE: struct ENTRY should really be in entry.c - to keep
   it private.  But it seems that fread() and fwrite() in
   entrybase.c needs direct access to the struct, a pointer
   to it gives run-time errors.
*/

typedef struct ENTRY {
  char number[NUMBERSIZE];
  char name[NAMESIZE];
  char address[ADDRESSSIZE];
  Category category;
  Status status;
} ENTRY, *Entry;

/* entryError: reports fatal entry errors, exits program */
void entryError(const char *report);

/* entryWarning: reports warnings */
int entryWarning(const char *report);
```

```
/* getName: returns new entry name from the keyboard */
char *getName();

/* getAddress: returns new entry address from the keyboard */
char *getAddress();

/* getCategory: returns new entry category from the keyboard */
Category getCategory();

/* getStatus: gets status from the keyboard */
Status getStatus();

/* newEntry: returns a blank entry */
Entry newEntry();

/* getEntry: returns a new entry from the keyboard */
Entry getEntry();

/* copyEntry: returns a duplicate of the given entry */
Entry copyEntry(const Entry entry);

/* setNumber: amends entry number */
Entry setNumber(const char *number, const Entry entry);

/* setName: amends entry name */
Entry setName(const char *name, const Entry entry);

/* setAddress: amends entry address */
Entry setAddress(const char *address, const Entry entry);

/* setCategory: amends entry category */
Entry setCategory(Category category, const Entry entry);

/* setStatus: amends entry status */
Entry setStatus(Status status, const Entry entry);

/* setEntry: returns a new entry with the given values */
Entry setEntry(const char *name, const char *address, Category cat);

/* entryNumber: returns number from entry */
char *entryNumber(const Entry entry);

/* entryName: returns name from entry */
char *entryName(const Entry entry);

/* entryAddress: returns address from entry */
char *entryAddress(const Entry entry);

/* entryCategory: returns category from entry */
Category entryCategory(const Entry entry);

/* entryStatus: returns status from entry */
Status entryStatus(const Entry entry);

/* categoryToString: returns category as a string */
char *categoryToString(Category category);

/* statusToString: returns status as a string */
char *statusToString(Status status);
```

```
/* equalEntries: returns 1 (true) if two entries have
   identical names and addresses */
int equalEntries(const Entry e1, const Entry e2);

/* printEntryHeadings: displays entry headings */
int printEntryHeadings();

/* printEntry: displays entry */
int printEntry(const Entry entry);

#endif
```

5.2 entry.c

The functions in *entry.c* are quite straightforward.

```
/* entry.c: a marathon entry implementation */
/* ver 1.6 26Jan2011 */

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

/* entryError: reports fatal entry errors, exits program */
void entryError(const char *report)
{
  printf("\n%s\n", report);
  exit(EXIT_FAILURE);
}

/* entryWarning: reports warnings */
int entryWarning(const char *report)
{
  printf("%s\n", report);
  return 0;
}

/* getName: returns new entry name from the keyboard */
char *getName()
{
  return getStr("Name?", NAMESIZE);
}

/* getAddress: returns new entry address from the keyboard */
char *getAddress()
{
  return getStr("Address?", ADDRESSSIZE);
}
```

```
/* getCategory: returns new entry category from the keyboard */
Category getCategory()
{
  char *category;

  category = getStr("Full or half marathon (F or H)?", 15);
  if (category[0] == 'f' || category[0] == 'F')
    return full;
  else if (category[0] == 'h' || category[0] == 'H')
    return half;
  else
    entryWarning("getCategory: unknown category");
  return unknownCategory;
}

/* getStatus: returns new entry status from the keyboard */
Status getStatus()
{
  char *status;

  status = getStr("Current or withdrawn (C or W)?", 15);
  if (status[0] == 'c' || status[0] == 'C')
    return current;
  else if (status[0] == 'w' || status[0] == 'W')
    return withdrawn;
  else
   entryWarning("getStatus: unknown status");
  return unknownStatus;
}

/* newEntry: returns a blank entry */
Entry newEntry()
{
  Entry entry;

  entry = malloc(sizeof(struct ENTRY));
  if (entry == NULL)
    entryError("newEntry: out of memory");
  strcpy(entry->number, "0");
  strcpy(entry->name, "");
  strcpy(entry->address, "");
  entry->category = unknownCategory;
  entry->status = unknownStatus;
  return entry;
}

/* getEntry: returns a new entry from the keyboard */
Entry getEntry()
{
  Entry entry = newEntry();

  strcpy(entry->number, "0");
  strcpy(entry->name, getName());
  strcpy(entry->address, getAddress());
  entry->category = getCategory();
  entry->status = current;
  return entry;
}
```

```
/* copyEntry: returns a duplicate of the given entry */
Entry copyEntry(const Entry entry)
{
  Entry entry2;

  entry2 = newEntry();
  strcpy(entry2->number, entry->number);
  strcpy(entry2->name, entry->name);
  strcpy(entry2->address, entry->address);
  entry2->category = entry->category;
  entry2->status = entry->status;
  return entry2;
}

/* setNumber: amends entry number */
Entry setNumber(const char *number, const Entry entry)
{
  int intNumber;
  Entry entry2;

  if ((strlen(number)) > MAXNUMBERLENGTH)
    entryWarning("setNumber: given number too large");
  intNumber = atoi(number);
  if (intNumber < MINENTRYNUMBER || intNumber > MAXENTRYNUMBER)
    entryWarning("setNumber: given number out of range");
  entry2 = copyEntry(entry);
  strcpy(entry2->number, number);
  return entry2;
}

/* setName: amends entry name */
Entry setName(const char *name, const Entry entry)
{
  Entry entry2;

  if (strlen(name) > NAMESIZE)
    entryWarning("setName: name too long");
    entry2 = copyEntry(entry);
  strncpy(entry2->name, name, NAMESIZE);
  entry2->name[NAMESIZE-1] = '\0';
  return entry2;
}

/* setAddress: amends entry address */
Entry setAddress(const char *address, const Entry entry)
{
  Entry entry2;

  if (strlen(address) > ADDRESSSIZE)
    entryWarning("setAddress: address too long");
  entry2 = copyEntry(entry);
  strncpy(entry2->address, address, ADDRESSSIZE);
  entry2->address[ADDRESSSIZE-1] = '\0';
  return entry2;
}
```

```
/* setCategory: amends entry category */
Entry setCategory(Category category, const Entry entry)
{
  Entry entry2;

  entry2 = copyEntry(entry);
  entry2->category = category;
  return entry2;
}

/* setStatus: amends entry status */
Entry setStatus(Status status, const Entry entry)
{
  Entry entry2;

  entry2 = copyEntry(entry);
  entry2->status = status;
  return entry2;
}

/* setEntry: returns a new entry with the given values */
Entry setEntry(const char *name, const char *address, Category cat)
{
  Entry entry;

  entry = newEntry();
  strcpy(entry->number, "0");
  entry = setName(name, entry);
  entry = setAddress(address, entry);
  entry = setCategory(cat, entry);
  entry = setStatus(current, entry);
  return entry;
}

/* entryNumber: returns number from entry */
char *entryNumber(const Entry entry)
{
  char *number;

  number = malloc(NUMBERSIZE);
  if (number == NULL)
    entryError("entryNumber: out of memory");
  strcpy(number, entry->number);
  return number;
}

/* entryName: returns name from entry */
char *entryName(const Entry entry)
{
  char *name;

  name = malloc(NAMESIZE);
  if (name == NULL)
    entryError("entryName: out of memory");
  strcpy(name, entry->name);
  return name;
}
```

```
/* entryAddress: returns address from entry */
char *entryAddress(const Entry entry)
{
  char *address;

  address = malloc(ADDRESSSIZE);
  if (address == NULL)
    entryError("entryAddress: out of memory");
  strcpy(address, entry->address);
  return address;
}

/* entryCategory: returns category from entry */
Category entryCategory(const Entry entry)
{
  return entry->category;
}

/* entryStatus: returns status from entry */
Status entryStatus(const Entry entry)
{
  return entry->status;
}

/* categoryToString: returns category as a string */
char *categoryToString(Category category)
{
  char *string;
  int size;

  size = strlen("unknown") + 1;
  string = malloc(size);
  if (category == half)
    strcpy(string, "half");
  else if (category == full)
    strcpy(string, "full");
  else
    strcpy(string, "unknown");
  return string;
}

/* statusToString: returns status as a string */
char *statusToString(Status status)
{
  char *string;
  int size;

  size = strlen("withdrawn") + 1;
  string = malloc(size);
  if (status == current)
    strcpy(string, "current");
  else if (status == withdrawn)
    strcpy(string, "withdrawn");
  else
    strcpy(string, "unknown");
  return string;
}

/* TO DO: equalEntries may be better written if it compared
   names, post codes and house numbers.  Need to break address
   into component parts. */
```

```
/* equalEntries: returns 1 (true) if two entries have
   identical names and addresses */
int equalEntries(const Entry e1, const Entry e2)
{
  if (equalStringsIgnoreCase(entryName(e1), entryName(e2)) &&
       equalStringsIgnoreCase(entryAddress(e1), entryAddress(e2)))
    return 1;
  else
    return 0;
}

/* printEntryHeadings: displays entry headings */
int printEntryHeadings()
{
  printf("%-6s %-15s %-23s %-10s %-10s\n",
          "Number", "Name", "Address", "Category", "Status");
  return 0;
}

/* printEntry: displays entry */
int printEntry(const Entry entry)
{
  printf("%6s %-15s %-23s %-10s %-10s\n",
          entryNumber(entry), entryName(entry),
          entryAddress(entry),
          categoryToString(entryCategory(entry)),
          statusToString(entryStatus(entry)));
  return 0;
}
```

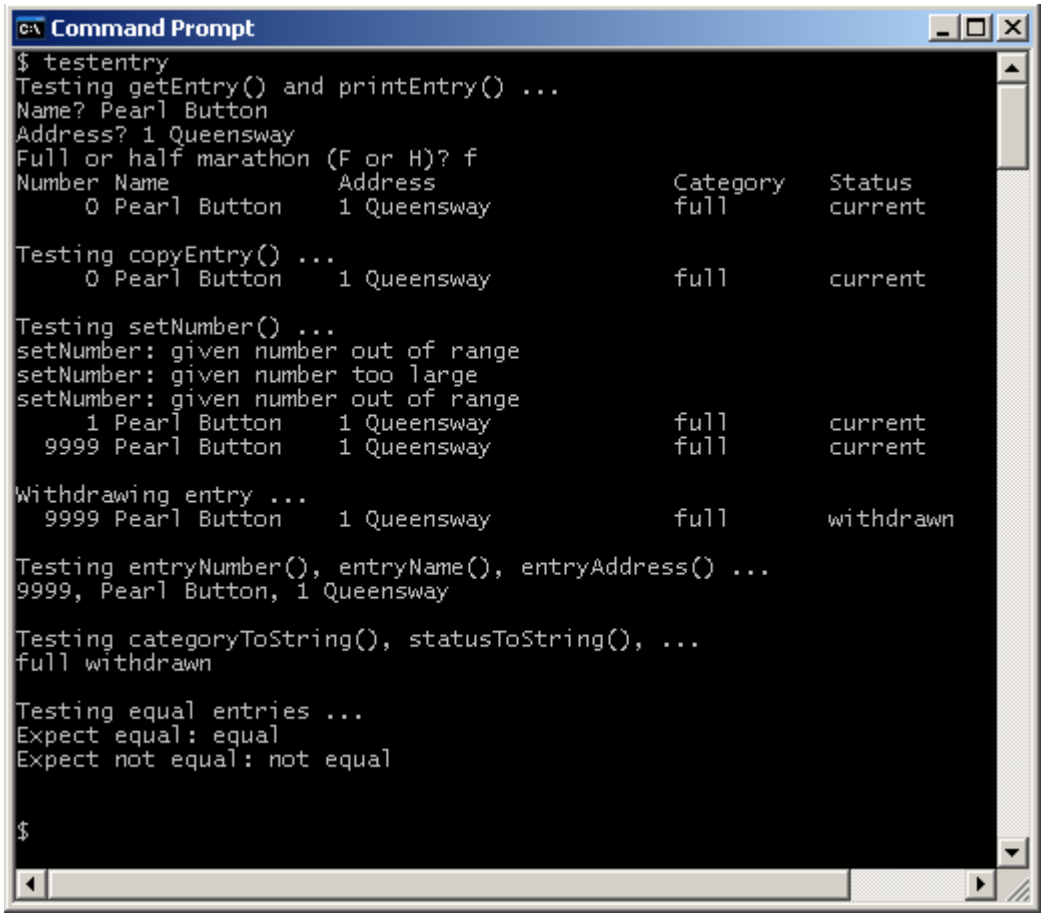
5.3 testentry.c

We aim to test all the functions ... With testing we state *what* we expect to see - if we do not get what we expected then we have made a mistake somewhere. If you test without any specific expectations then you are just playing.

```
/* testentry.c: tests marathon entry */
/* ver 1.1 9Jan2011 */

#include <stdio.h>
#include "entry.h"

int main()
{
  Entry e1, e2;

  printf("Testing getEntry() and printEntry() ... \n");
  e1 = getEntry();
  printEntryHeadings();
  printEntry(e1);
  printf("\n");



  printf("Testing copyEntry() ... \n");
  e2 = copyEntry(e1);
  printEntry(e2);
  printf("\n");
```

```
    printf("Testing setNumber() ... \n");
    e2 = setNumber("-1", e2);
    e2 = setNumber("10000", e2);
    e2 = setNumber("1", e2);
    printEntry(e2);
    e2 = setNumber("9999", e2);
    printEntry(e2);
    printf("\n");

    printf("Withdrawing entry ... \n");
    e2 = setStatus(withdrawn, e2);
    printEntry(e2);
    printf("\n");

    printf("Testing entryNumber(), entryName(), entryAddress()...\n");
    printf("%s, %s, %s\n",
             entryNumber(e2), entryName(e2), entryAddress(e2));
    printf("\n");

    printf("Testing categoryToString(), statusToString(), ...\n");
    printf("%s %s\n", categoryToString(entryCategory(e2)),
                      statusToString(entryStatus(e2)));
    printf("\n");

    printf("Testing equal entries ... \n");
    printf("Expect equal: ");
    equalEntries(e1, e2)? printf("equal\n"): printf("not equal\n");
    e2 = setName("Jo King", e2);
    printf("Expect not equal: ");
    equalEntries(e1, e2)? printf("equal\n"): printf("not equal\n");

    printf("\n");
    return 0;
}
```

I wrote and tested entry as one complete file before splitting it into three parts: one for the interface, one for the implementation and one for testing.



5.4 entrybase.h

entrybase maintains a file of entries.

```
/* entrybase.h: interface for entrybase.c */
/* ver 1.2 21Jan2011 */

#ifndef ENTRYBASE_H
#define ENTRYBASE_H

#include "entry.h"

/* entryFileError: reports fatal file errors, exits program */
void entryFileError(const char *report);

/* entryFileWarning: displays a warning, waits for user
   to acknowledge */
int entryFileWarning(const char *report);

/* entryFileExists: returns 1 (true) if entry file exists, 0
otherwise */
int entryFileExists(const char *fileName);

/* newEntryFile: creates a new empty file with the given
   external file name */
int newEntryFile(const char *fileName);
```

```
/* addEntry: appends the given entry to the file */
int addEntry(const Entry entry, const char *fileName);

/* containsEntryNumber: returns 1 (true) if a competitor in
   the file has the given competitor number */
int containsEntryNumber(const char *number, const char *fileName);

/* getEntryFromFile: returns the entry with the given number. */
Entry getEntryFromFile(const char *number, const char *fileName);

/* getNameFromFile: returns the name for the given number */
char *getNameFromFile(const char *number, const char *fileName);

/* getCategoryFromFile: returns the category for the given number */
Category categoryFromFile(const char *number, const char *fileName);

/* updateEntryFile: updates entry file with the given entry */
int updateEntryFile(const char *compNum, const Entry newEntry,
                    const char *fileName);

/* amendEntryStatus: changes entry status for the given status */
int amendEntryStatus(const char *compNum, Status status,
                     const char *fileName);

/* printEntryFile: displays contents of the file */
int printEntryFile(const char *fileName);

#endif
```

5.5 entrybase.c

entrybase maintains a file of entries. It adds entries to the file, updates the file by amending entries, and prints the entries in the file.

```
/* entrybase.c: maintains file of entries */
/* ver 1.4 26Jan2011 */

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

/* entryFileError: reports fatal file errors, exits program */
void entryFileError(const char *report)
{
  printf("\n%s\n", report);
  exit(EXIT_FAILURE);
}

/* entryFileWarning: displays a warning, waits for user to
   acknowledge */
int entryFileWarning(const char *report)
{
  printf("\n%s\n", report);
  getStr("Press return to continue ... ", 5);
  return 0;
}
```

```
/* entryFileExists: returns 1 (true) if entry file exists, 0
otherwise */
int entryFileExists(const char *fileName)
{
  FILE *file;

  file = fopen(fileName, "rb");
  if (file != NULL) {
    fclose(file);
    return 1;
  }
  return 0;
}

/* NOTE: File handling errors are spread over several functions.
   Could file handling be confined to just one function?  But
   file handling for each function enables specific error
   messages to be created.  Perhaps best left as it is. */

/* newEntryFile: creates a new empty file with the given
   external file name */
int newEntryFile(const char *fileName)
{
  FILE *file;

  if (entryFileExists(fileName))
    return entryFileWarning("newEntryFile: file already exists");
  file = fopen(fileName, "wb");
  if (file == NULL)
    entryFileError("newEntryFile: cannot open file");
  fclose(file);
  return 0;
}

/* NOTE: It seems that fread() and fwrite() both need direct
   access to the entry struct - a pointer to the struct generates
   run time errors. */

/* nextCompetitorNumber: returns the next available
   competitor number */
char *nextCompetitorNumber(const char *fileName)
{
  int number;
  FILE *file;
  ENTRY entry;

  number = MINENTRYNUMBER;
  file = fopen(fileName, "rb");
  if (file == NULL)
    entryFileError("nextCompetitorNumber: cannot open file");
  fread(&entry, sizeof(ENTRY), 1, file);
  while (!feof(file)) {
    number++;
    fread(&entry, sizeof(ENTRY), 1, file);
  }
  fclose(file);
  if (number > MAXENTRYNUMBER)
    entryFileWarning("nextCompetitorNumber: limit reached");
  return intToString(number);
}
```

```
/* duplicateEntry: returns 1 (true) if an entry in the file
   matches the given entry, 0 otherwise */
int duplicateEntry(const Entry e, const char *fileName)
{
  FILE *file;
  ENTRY entry;

  if (!entryFileExists(fileName))
    entryFileError("duplicateEntry: file does not exist");
  file = fopen(fileName, "rb");
  if (file == NULL)
    entryFileError("duplicateEntry: cannot open file");
  fread(&entry, sizeof(ENTRY), 1, file);
  while (!feof(file)) {
    if (equalEntries(e, &entry)) {
      fclose(file);
      return 1;
    }
    fread(&entry, sizeof(ENTRY), 1, file);
  }
  fclose(file);
  return 0;
}

/* addEntry: appends the given entry to the file */
int addEntry(const Entry entry, const char *fileName)
{
  FILE *file;
  ENTRY e;
  int result;
  char *number;

  if (duplicateEntry(entry, fileName)) {
    entryFileWarning("addEntry: duplicate entry");
    return 1;
  }
  number = nextCompetitorNumber(fileName);
  e = *setNumber(number, entry);
  file = fopen(fileName, "ab");
  if (file == NULL)
    entryFileError("addEntry: cannot open file");
  result = fwrite(&e, sizeof(ENTRY), 1, file);
  if (result != 1)
    entryFileError("addEntry: cannot add entry");
  fclose(file);
  return 0;
}
```

```
/* containsEntryNumber: returns 1 (true) if a competitor in
   the file has the given competitor number */
int containsEntryNumber(const char *number, const char *fileName)
{
  FILE *file;
  ENTRY entry;

  file = fopen(fileName, "rb");
  if (file == NULL)
    entryFileError("containsEntryNumber: cannot open file");
  fread(&entry, sizeof(ENTRY), 1, file);
  while (!feof(file)) {
    if (strcmp(number, entryNumber(&entry)) == 0) {
      fclose(file);
      return 1;
    }
    fread(&entry, sizeof(ENTRY), 1, file);
  }
  fclose(file);
  return 0;
}

/* getEntryFromFile: returns the entry with the given number. */
Entry getEntryFromFile(const char *number, const char *fileName)
{
  FILE *file;
  ENTRY entry;

  if (!containsEntryNumber(number, fileName)) {
    entryFileWarning(
      "getEntryFromFile: no entry with given number found");
    return newEntry();
  }
  file = fopen(fileName, "rb");
  if (file == NULL)
    entryFileError("getEntryFromFile: cannot open file");
  fread(&entry, sizeof(ENTRY), 1, file);
  while (!feof(file)) {
    if (strcmp(number, entryNumber(&entry)) == 0) {
      fclose(file);
      return copyEntry(&entry);
    }
   fread(&entry, sizeof(ENTRY), 1, file);
  }
  return newEntry();
}

/* getNameFromFile: returns the name for the given number */
char *getNameFromFile(const char *number, const char *fileName)
{
  Entry entry;

  entry = getEntryFromFile(number, fileName);
  return entryName(entry);
}
```

```
/* getCategoryFromFile: returns the category for the given entry */
Category categoryFromFile(const char *number, const char *fileName)
{
  Entry entry;
  entry = getEntryFromFile(number, fileName);
  return entryCategory(entry);
}

/* getLocationInFile: returns record number for the given
   competitor number.  Returns -1 if record not found */
int getLocationInFile(const char *number, const char *fileName)
{
  FILE *file;
  ENTRY entry;
  int n = 0;

  if (!containsEntryNumber(number, fileName)) {
    entryFileWarning(
            "getLocationInFile: no entry with given number found");
    return -1;
  }
  file = fopen(fileName, "rb");
  if (file == NULL)
    entryFileError("getLocationInFile: cannot open file");
  fread(&entry, sizeof(ENTRY), 1, file);
  while (!feof(file)) {
    if (strcmp(number, entryNumber(&entry)) == 0) {
      fclose(file);
      return n;
    }
    fread(&entry, sizeof(ENTRY), 1, file);
    n++;
  }
  return -1;
}

/* updateEntryFile: updates entry file with the given entry */
int updateEntryFile(const char *compNum, const Entry entry,
                    const char *fileName)
{
  FILE *file;
  ENTRY e;
  int location;
  int result;

  if (!entryFileExists(fileName))
    entryFileError("updateEntryfile: cannot find entry file");
  location = getLocationInFile(compNum, fileName);
  file = fopen(fileName, "rb+");
  if (file == NULL)
    entryFileError("updateEntryFile: cannot open entry file");
  result = fseek(file, location * sizeof(ENTRY), SEEK_SET);
  if (result != 0) {
    fclose(file);
    entryFileError("updateEntryFile: positioning error");
  }
  e = *copyEntry(entry);
  fwrite(&e, sizeof(ENTRY), 1, file);
  fclose(file);
  return 0;
}
```

```
/* amendEntryStatus: changes entry status for the given status */
int amendEntryStatus(const char *compNum, Status status,
                     const char *fileName)
{
  ENTRY entry;

  entry = *getEntryFromFile(compNum, fileName);
  if (strcmp(entryNumber(&entry), "0") == 0)
    return 1;
  entry = *setStatus(status, &entry);
  updateEntryFile(compNum, &entry, fileName);
  return 0;
}

/* TO DO: format the printing of entries into pages
   with page numbers, headings, totals, ... */

/* printEntryFile: displays contents of entry file */
int printEntryFile(const char *fileName)
{
  FILE *file;
  ENTRY entry;

  if (!entryFileExists(fileName))
    entryFileError("printEntryFile: file does not exist");
  file = fopen(fileName, "rb");
  if (file == NULL)
    entryFileError("printEntryFile: cannot open file");
  printEntryHeadings();
  fread(&entry, sizeof(ENTRY), 1, file);
  while (!feof(file)) {
    printEntry(&entry);
    fread(&entry, sizeof(ENTRY), 1, file);
  }
  fclose(file);
  return 0;
}
```

5.6 testentrybase.c

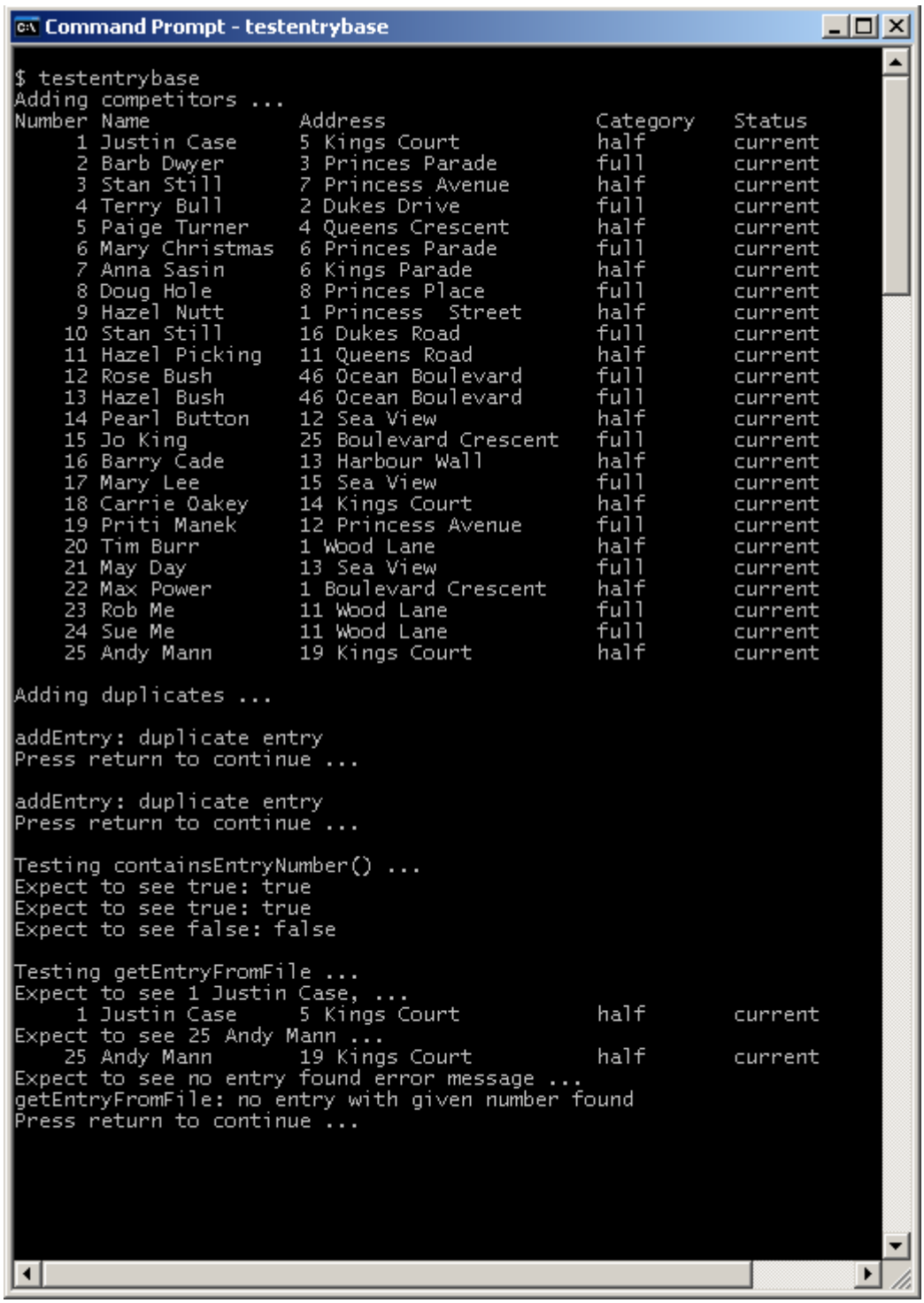
We write a *populate()* function that adds about 25 entries to the file for testing purposes. Just imagine how tedious it would be to type in details for 25 competitors every time you test ran the program. The filename is hard-wired into the program - we allow the user to enter their chosen filenames later on.

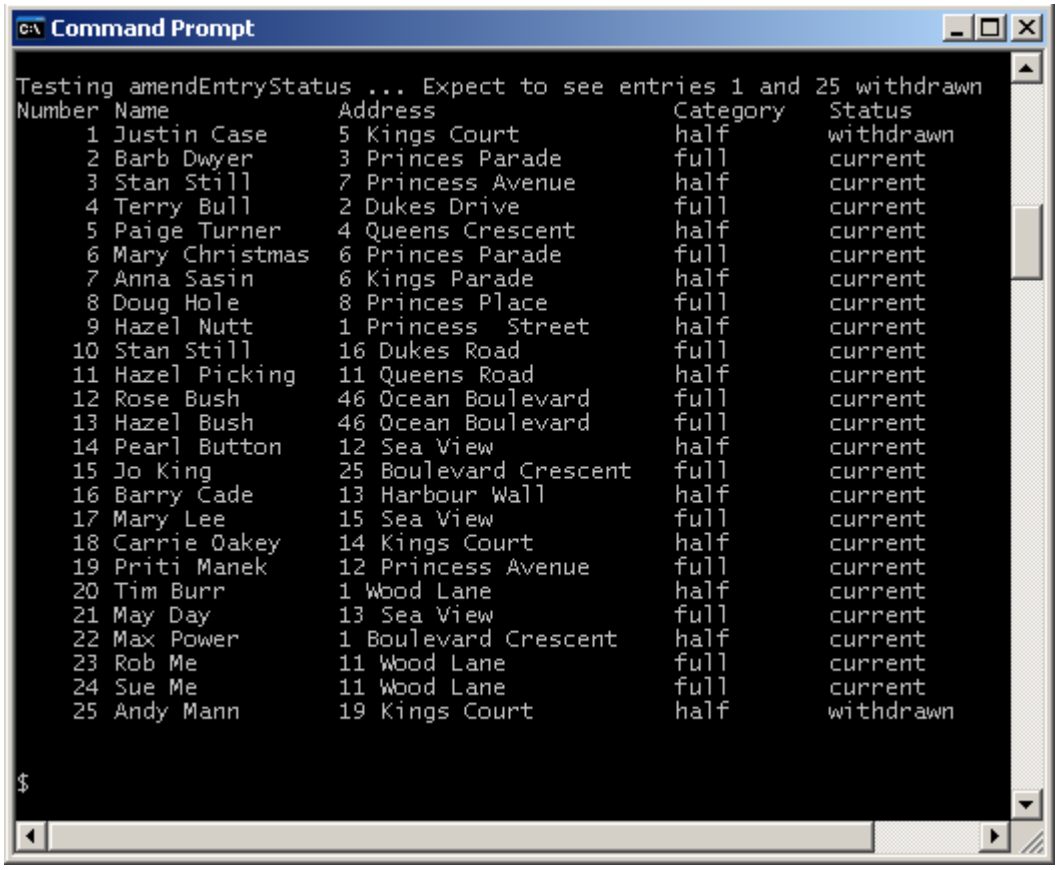
```
/* testentrybase.c: tests entry base */
/* ver 1.0 11Jan2011 */

#include <stdio.h>
#include <stdlib.h>
#include "entry.h"
#include "entrybase.h"

char *fileName = "entries2010.dat";
```

```
int populate()
{
  Entry e;

  e = setEntry("Justin Case", "5 Kings Court", half);
  addEntry(e, fileName);
  e = setEntry("Barb Dwyer", "3 Princes Parade", full);
  addEntry(e, fileName);
  e = setEntry("Stan Still", "7 Princess Avenue", half);
  addEntry(e, fileName);
  e = setEntry("Terry Bull", "2 Dukes Drive", full);
  addEntry(e, fileName);
  e = setEntry("Paige Turner", "4 Queens Crescent", half);
  addEntry(e, fileName);
  e = setEntry("Mary Christmas", "6 Princes Parade", full);
  addEntry(e, fileName);
  e = setEntry("Anna Sasin", "6 Kings Parade", half);
  addEntry(e, fileName);
  e = setEntry("Doug Hole", "8 Princes Place", full);
  addEntry(e, fileName);
  e = setEntry("Hazel Nutt", "1 Princess  Street", half);
  addEntry(e, fileName);
  e = setEntry("Stan Still", "16 Dukes Road", full);
  addEntry(e, fileName);
  e = setEntry("Hazel Picking", "11 Queens Road", half);
  addEntry(e, fileName);
  e = setEntry("Rose Bush", "46 Ocean Boulevard", full);
  addEntry(e, fileName);
  e = setEntry("Hazel Bush", "46 Ocean Boulevard", full);
  addEntry(e, fileName);
  e = setEntry("Pearl Button", "12 Sea View", half);
  addEntry(e, fileName);
  e = setEntry("Jo King", "25 Boulevard Crescent", full);
  addEntry(e, fileName);
  e = setEntry("Barry Cade", "13 Harbour Wall", half);
  addEntry(e, fileName);
  e = setEntry("Mary Lee", "15 Sea View", full);
  addEntry(e, fileName);
  e = setEntry("Carrie Oakey", "14 Kings Court", half);
  addEntry(e, fileName);
  e = setEntry("Priti Manek", "12 Princess Avenue", full);
  addEntry(e, fileName);
  e = setEntry("Tim Burr", "1 Wood Lane", half);
  addEntry(e, fileName);
  e = setEntry("May Day", "13 Sea View", full);
  addEntry(e, fileName);
  e = setEntry("Max Power", "1 Boulevard Crescent", half);
  addEntry(e, fileName);
  e = setEntry("Rob Me", "11 Wood Lane", full);
  addEntry(e, fileName);
  e = setEntry("Sue Me", "11 Wood Lane", full);
  addEntry(e, fileName);
  e = setEntry("Andy Mann", "19 Kings Court", half);
  addEntry(e, fileName);
  return 0;
}
```

```
int main()
{
  Entry e;

  if (!entryFileExists(fileName))
    newEntryFile(fileName);
  printf("Adding competitors ... \n");
  populate();
  printEntryFile(fileName);
  printf("\n");

  printf("Adding duplicates ... \n");
  e = setEntry("Andy Mann", "19 Kings Court", full);
  addEntry(e, fileName);
  e = setEntry("Justin Case", "5 Kings Court", half);
  addEntry(e, fileName);
  printf("\n");

  printf("Testing containsEntryNumber() ... \n");
  printf("Expect to see true: ");
  containsEntryNumber("1", fileName)?
      printf("true\n"): printf("false\n");
  printf("Expect to see true: ");
  containsEntryNumber("25", fileName)?
      printf("true\n"): printf("false\n");
  printf("Expect to see false: ");
  containsEntryNumber("26", fileName)?
      printf("true\n"): printf("false\n");
  printf("\n");

  printf("Testing getEntryFromFile ... \n");
  printf("Expect to see 1 Justin Case, ... \n");
  printEntry(getEntryFromFile("1", fileName));
  printf("Expect to see 25 Andy Mann ... \n");
  printEntry(getEntryFromFile("25", fileName));
  printf("Expect to see no entry found error message ... ");
  getEntryFromFile("26", fileName);
  printf("\n");

  printf("Testing amendEntryStatus ... ");
  printf("Expect to see entries 1 and 25 withdrawn\n");
  amendEntryStatus("1", withdrawn, fileName);
  amendEntryStatus("25", withdrawn, fileName);
  printEntryFile(fileName);

  printf("\n");
  return 0;
}
```

We make the testing rather thorough because this module is the basis of the rest of the program. If this module is wrong there is no chance of the modules that follow being right.


Command Prompt - testentrybase
$ testentrybase
Adding competitors ...
Number Name Address Category Status
1 Justin Case 5 Kings Court half current
2 Barb Dwyer 3 Princes Parade full current
3 Stan Still 7 Princess Avenue half current
4 Terry Bull 2 Dukes Drive full current
5 Paige Turner 4 Queens Crescent half current
6 Mary Christmas 6 Princes Parade full current
7 Anna Sasin 6 Kings Parade half current
8 Doug Hole 8 Princes Place full current
9 Hazel Nutt 1 Princess Street half current
10 Stan Still 16 Dukes Road full current
11 Hazel Picking 11 Queens Road half current
12 Rose Bush 46 Ocean Boulevard full current
13 Hazel Bush 46 Ocean Boulevard full current
14 Pearl Button 12 Sea View half current
15 Jo King 25 Boulevard Crescent full current
16 Barry Cade 13 Harbour Wall half current
17 Mary Lee 15 Sea View full current
18 Carrie Oakey 14 Kings Court half current
19 Priti Manek 12 Princess Avenue full current
20 Tim Burr 1 Wood Lane half current
21 May Day 13 Sea View full current
22 Max Power 1 Boulevard Crescent half current
23 Rob Me 11 Wood Lane full current
24 Sue Me 11 Wood Lane full current
25 Andy Mann 19 Kings Court half current
Adding duplicates ...
addEntry: duplicate entry
Press return to continue ...
addEntry: duplicate entry
Press return to continue ...
Testing containsEntryNumber() ...
Expect to see true: true
Expect to see true: true
Expect to see false: false
Testing getEntryFromFile ...
Expect to see 1 Justin Case, ...
1 Justin Case 5 Kings Court half current
Expect to see 25 Andy Mann ...
25 Andy Mann 19 Kings Court half current
Expect to see no entry found error message ...
getEntryFromFile: no entry with given number found
Press return to continue ...


Command Prompt
Testing amendEntryStatus ... Expect to see entries 1 and 25 withdrawn
Number Name Address Category Status
1 Justin Case 5 Kings Court half withdrawn
2 Barb Dwyer 3 Princes Parade full current
3 Stan Still 7 Princess Avenue half current
4 Terry Bull 2 Dukes Drive full current
5 Paige Turner 4 Queens Crescent half current
6 Mary Christmas 6 Princes Parade full current
7 Anna Sasin 6 Kings Parade half current
8 Doug Hole 8 Princes Place full current
9 Hazel Nutt 1 Princess Street half current
10 Stan Still 16 Dukes Road full current
11 Hazel Picking 11 Queens Road half current
12 Rose Bush 46 Ocean Boulevard full current
13 Hazel Bush 46 Ocean Boulevard full current
14 Pearl Button 12 Sea View half current
15 Jo King 25 Boulevard Crescent full current
16 Barry Cade 13 Harbour Wall half current
17 Mary Lee 15 Sea View full current
18 Carrie Oakey 14 Kings Court half current
19 Priti Manek 12 Princess Avenue full current
20 Tim Burr 1 Wood Lane half current
21 May Day 13 Sea View full current
22 Max Power 1 Boulevard Crescent half current
23 Rob Me 11 Wood Lane full current
24 Sue Me 11 Wood Lane full current
25 Andy Mann 19 Kings Court half withdrawn
$