Programming Project

Terry Marris January 2010

6 Runner

In runner we are concerned with recording each runner's times.

6.1 runner.h

A runner has a number, a start time, a finish time, a race time and a status. A status is either *dns* for did not start, *dnf* for did not finish, or *fin* for finished.

```
/* runner.h: runner interface */
/* ver 1.0 17Jan2011 */

#ifndef RUNNER_H
#define RUNNER_H

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

#define MINRACETIME 3000

typedef struct tm Time;

typedef enum { dns, dnf, fin } RunnerStatus;

typedef struct RUNNER {
  char number[NUMBERSIZE];
  Time startTime;
  Time finishTime;
  int raceTime;
  RunnerStatus status;
} RUNNER, *Runner;

/* runnerError: reports fatal errors, halts program */
void runnerError(const char *report);

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

/* defaultTime: returns 1 Jan 0:00:00 */
Time defaultTime();

/* setTime: returns time from given hr, min, sec */
Time setTime(int hr, int min, int sec);

/* newRunner: returns a new runner */
Runner newRunner();

/* copyRunner: returns a duplicate of the given runner */
Runner copyRunner(const Runner runner);
```

```
/* equalRunners: returns 1 (true) if the given runners have
   the same runner number, 0 otherwise */
int equalRunners(const Runner r1, const Runner r2);

/* setRunnerNumber: amends runner number */
Runner setRunnerNumber(const char *number, const Runner runner,
                       const char *entryFileName);

/* raceTime: returns the race time in seconds */
int raceTime(Time startTime, Time finishTime,
             const char *number, const char *entryFileName);

/* setStartTime: sets runner's start time */
Runner setStartTime(Time startTime, const Runner runner);

/* setFinishTime: sets runner's finish time */
Runner setFinishTime(Time finishTime, const Runner runner, char
*entryFileName);

/* raceTimeToString: returns race time in the format hrs:min:sec */
char *raceTimeToString(int seconds);

/* runnerNumber: returns number from runner */
char *runnerNumber(const Runner runner);

/* runnerStartTime: returns start time from runner */
Time runnerStartTime(const Runner runner);

/* runnerFinishTime: returns finish time from runner */
Time runnerFinishTime(const Runner runner);

/* runnerStatus: returns status from runner */
RunnerStatus runnerStatus(const Runner runner);

/* runnerStatusToString: returns status as string */
char *runnerStatusToString(RunnerStatus status);

/* runnerRaceTime: returns runners race time in seconds */
int runnerRaceTime(const Runner runner);

/* printRunnerHeadings: displays headings */
int printRunnerHeadings();

/* printRunner: displays runner */
int printRunner(const Runner runner);

#endif
```

6.2 runner.c

We see that runner is primarily concerned with times. Perhaps we could/should have a separate module just for processing times?

```
/* runner.c: a runner has a number, a start time and
   (hopefully) a finish time */
/* ver 1.0  17Jan2011 */

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

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

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

/* defaultTime: returns 1 Jan 0:00:00 */
Time defaultTime()
{
  Time time;

  time.tm_mday = 1;
  time.tm_wday = 0;
  time.tm_yday = 0;
  time.tm_mon = 0;
  time.tm_year = 2010 - 1900;
  time.tm_hour = 0;
  time.tm_min = 0;
  time.tm_sec = 0;
  time.tm_isdst = -1;
  if (-1 == mktime(&time))
    runnerWarning("defaultTime: invalid time");
  return time;
}
```

```
/* setTime: returns time from given hr, min, sec */
Time setTime(int hr, int min, int sec)
{
  Time time;

  time = defaultTime();
  time.tm_hour = hr;
  time.tm_min = min;
  time.tm_sec = sec;
  if (-1 == mktime(&time))
    runnerError("setTime: invalid time");
  return time;
}

/* newRunner: returns a new runner */
Runner newRunner()
{
  Runner runner;

  runner = malloc(sizeof(RUNNER));
  if (runner == NULL)
    runnerError("newRunner: out of memory");
  strcpy(runner->number, "0");
  runner->startTime = defaultTime();
  runner->finishTime = defaultTime();
  runner->raceTime = 0;
  runner->status = dns;
  return runner;
}

/* copyRunner: returns a duplicate of the given runner */
Runner copyRunner(const Runner runner)
{
  Runner r;

  if (runner == NULL) {
    runnerWarning("copyRunner: cannot copy a null runner");
    return newRunner();
  }
  r = newRunner();
  strcpy(r->number, runner->number);
  r->startTime = runner->startTime;
  r->finishTime = runner->finishTime;
  r->raceTime = runner->raceTime;
  r->status = runner->status;
  return r;
}

/* equalRunners: returns 1 (true) if the given runners have
   the same runner number, 0 otherwise */
int equalRunners(const Runner r1, const Runner r2)
{
  if (r1 == NULL || r2 == NULL)
    return runnerWarning("equalRunners: cannot compare NULL runner");
  if (strcmp(runnerNumber(r1), runnerNumber(r2)) == 0)
    return 1;
  else
    return 0;
}
```

```
/* setRunnerNumber: amends runner number */
Runner setRunnerNumber(const char *number, const Runner runner,
                       const char *entryFileName)
{
  Runner r;

  if (!containsEntryNumber(number, entryFileName)) {
    runnerWarning(
            "setRunnerNumber: runner number not in file of entries");
    return newRunner();
  }
  r = copyRunner(runner);
  strcpy(r->number, number);
  return r;
}

/* NOTE: what if runner has withdrawn?
*/

/* calendarTime: returns calendar time from local time */
time_t calendarTime(Time time)
{
  time_t calTime;

  calTime = mktime(&time);
  if (calTime == -1)
    runnerError("calendarTime: invalid date or time");
  return calTime;
}

/* raceTime: returns the race time in seconds */
int raceTime(Time startTime, Time finishTime,
             const char *number, const char *entryFileName)
{
  time_t start, finish;
  int time;
  Category category;

  if (!containsEntryNumber(number, entryFileName))
    return runnerWarning(
             "raceTime: runner number not in file of entries");
  category = categoryFromFile(number, entryFileName);
  if (category == unknownCategory)
    return runnerWarning("raceTime: runner category unknown");
  start = calendarTime(startTime);
  finish = calendarTime(finishTime);
  time = (int)difftime(finish, start);
  if ((category == half && time < MINRACETIME) ||
       (category == full && time < 2 * MINRACETIME))
    return runnerWarning("raceTime: race time too short");
  return time ;
}
```

```
/* setStartTime: sets runner's start time */
Runner setStartTime(Time startTime, const Runner runner)
{
  Runner r;

  r = copyRunner(runner);
  r->startTime = startTime;
  r->status = dnf;
  return r;
}

/* setFinishTime: sets runner's finish time */
Runner setFinishTime(Time finishTime, const Runner runner, char
*entryFileName)
{
  Runner r;

  if (runner->status != dnf) {
    runnerWarning("setFinishTime: no start time set");
    return runner;
  }
  r = copyRunner(runner);
  r->finishTime = finishTime;
  r->raceTime = raceTime(r->startTime, r->finishTime,
                         runnerNumber(r), entryFileName);
  r->status = fin;
  return r;
}

/* formatTime: returns time in the format hr:min:sec */
char *formatTime(int hr, int min, int sec)
{
  char *timeString;

  timeString = malloc(10 * sizeof(char));
  if (timeString == NULL)
    runnerError("formatTime: out of memory");
  strcpy(timeString, intToString(hr));
  strcat(timeString, ":");
  if (min < 10)
    strcat(timeString, "0");
  strcat(timeString, intToString(min));
  strcat(timeString, ":");
  if (sec < 10)
    strcat(timeString, "0");
  strcat(timeString, intToString(sec));
  return timeString;
}

/* timeToString: returns time in the format hrs:min:sec */
char *timeToString(Time time)
{
  int hr, min, sec;

  hr = time.tm_hour;
  min = time.tm_min;
  sec = time.tm_sec;
  return formatTime(hr, min, sec);
}
```

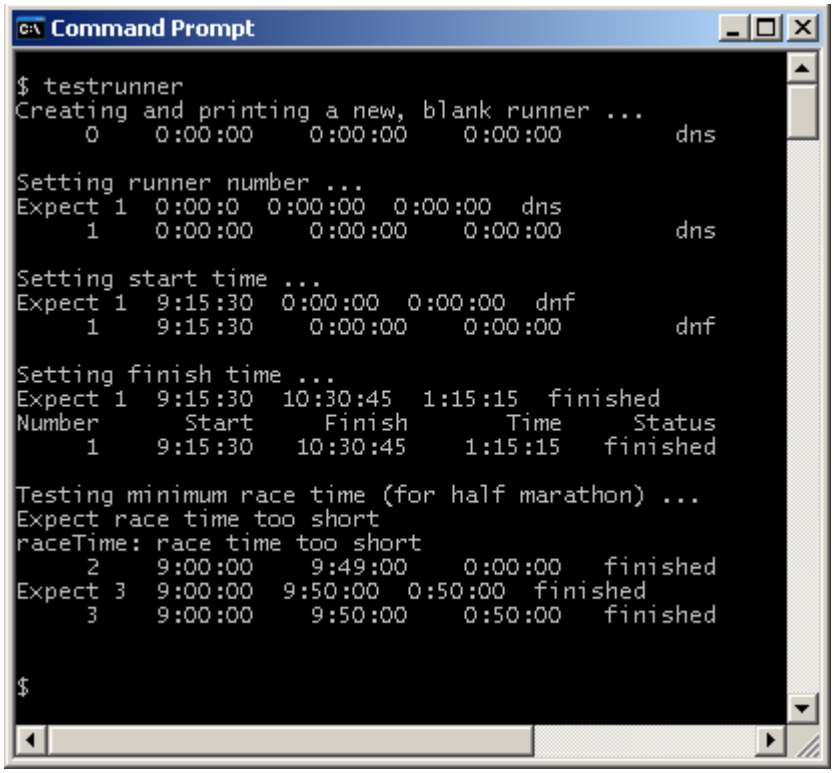
```
/* raceTimeToString: returns race time in the format hrs:min:sec */
char *raceTimeToString(int seconds)
{
  int hr, min, sec;

  hr = seconds / 3600;
  seconds = seconds % 3600;
  min = seconds / 60;
  seconds = seconds % 60;
  sec = seconds;
  return formatTime(hr, min, sec);
}

/* runnerNumber: returns number from runner */
char *runnerNumber(const Runner runner)
{
  char *number;

  number = malloc(NUMBERSIZE);
  if (number == NULL)
    runnerError("runnerNumber: out of memory");
  strcpy(number, runner->number);
  return number;
}

/* runnerStartTime: returns start time from runner */
Time runnerStartTime(const Runner runner)
{
  return runner->startTime;
}

/* runnerFinishTime: returns finish time from runner */
Time runnerFinishTime(const Runner runner)
{
  return runner->finishTime;
}

/* runnerStatus: returns status from runner */
RunnerStatus runnerStatus(const Runner runner)
{
  return runner->status;
}

/* runnerStatusToString: returns status as string */
char *runnerStatusToString(RunnerStatus status)
{
  char *string;
  int size;

  size = strlen("finished") + 1;
  string = malloc(size);
  if (status == dns)
    strcpy(string, "dns");
  else if (status == dnf)
    strcpy(string, "dnf");
  else if (status == fin)
    strcpy(string, "finished");
  else
    strcpy(string, "dns");
  return string;
}
```

```
/* runnerRaceTime: returns runners race time in seconds */
int runnerRaceTime(const Runner runner)
{
  return runner->raceTime;
}

/* printRunnerHeadings: displays headings */
int printRunnerHeadings()
{
  printf("%-6s %10s %10s %10s %10s\n",
          "Number", "Start", "Finish", "Time", "Status");
  return 0;
}

/* printRunner: displays runner */
int printRunner(const Runner runner)
{
  printf("%6s %10s %10s %10s %10s\n",
         runnerNumber(runner),
         timeToString(runnerStartTime(runner)),
         timeToString(runnerFinishTime(runner)),
         raceTimeToString(runnerRaceTime(runner)),
         runnerStatusToString(runnerStatus(runner)));
  return 0;
}
```

6.3 testrunner.c

```
/* testrunner.c: tests runner.c */
/* ver 1.0  17Jan2011 */

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

char *entryFileName = "entries2010.dat";

int main()
{
  Runner runner;

  printf("Creating and printing a new, blank runner ...\n");
  runner = newRunner();
  printRunner(runner);
  printf("\n");

  printf("Setting runner number ... \n");
  runner = setRunnerNumber("1", runner, entryFileName);
  printf("Expect 1  0:00:0  0:00:00  0:00:00  dns\n");
  printRunner(runner);
  printf("\n");

  printf("Setting start time ... \n");
  printf("Expect 1  9:15:30  0:00:00  0:00:00  dnf\n");
  runner = setStartTime(setTime(9, 15, 30), runner);
  printRunner(runner);
  printf("\n");
```

```
  printf("Setting finish time ... \n");
  runner = setFinishTime(setTime(10, 30, 45), runner, entryFileName);
  printf("Expect 1  9:15:30  10:30:45  1:15:15  finished\n");
  printRunnerHeadings();
  printRunner(runner);
  printf("\n");

  printf("Testing minimum race time (for half marathon) ... \n");
  printf("Expect race time too short\n");
  runner = newRunner();
  runner = setRunnerNumber("2", runner, entryFileName);
  runner = setStartTime(setTime(9, 0, 0), runner);
  runner = setFinishTime(setTime(9, 49, 0), runner, entryFileName);
  printRunner(runner);
  runner = newRunner();
  runner = setRunnerNumber("3", runner, entryFileName);
  printf("Expect 3  9:00:00  9:50:00  0:50:00  finished\n");
  runner = setStartTime(setTime(9, 0, 0), runner);
  runner = setFinishTime(setTime(9, 50, 0), runner, entryFileName);
  printRunner(runner);
  printf("\n");
  return 0;
}
```



6.4 runnerbase.h

runnerbase maintains a file of runners. A runner must be in the file of entries.

We see that file error and file warning functions are in runnerbase also. Perhaps there could/should be an error reporting module.

```
/* runnerbase.h: runnerbase interface */
/* ver 1.0  20Jan2011 */

#ifndef RUNNERBASE_H
#define RUNNERBASE_H

#include "runner.h"

/* runnerFileError: reports fatal errors, exits program */
void runnerFileError(char *report);

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

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

/* duplicateRunner: returns 1 (true) if a runner in the file
   has the given number, 0 otherwise */
int duplicateRunner(Runner r, char *fileName);

/* lastRaceTime: retrieves the last runner's race time
   from the runners file */
int lastRaceTime(char *fileName);

/* addRunner: appends the given runner to the file */
int addRunner(Runner runner, char *runnerFName, char *entryFName);

/* printRunnerFile: displays contents of runner file */
int printRunnerFile(char *fileName);

#endif
```

6.5 runnerbase.c

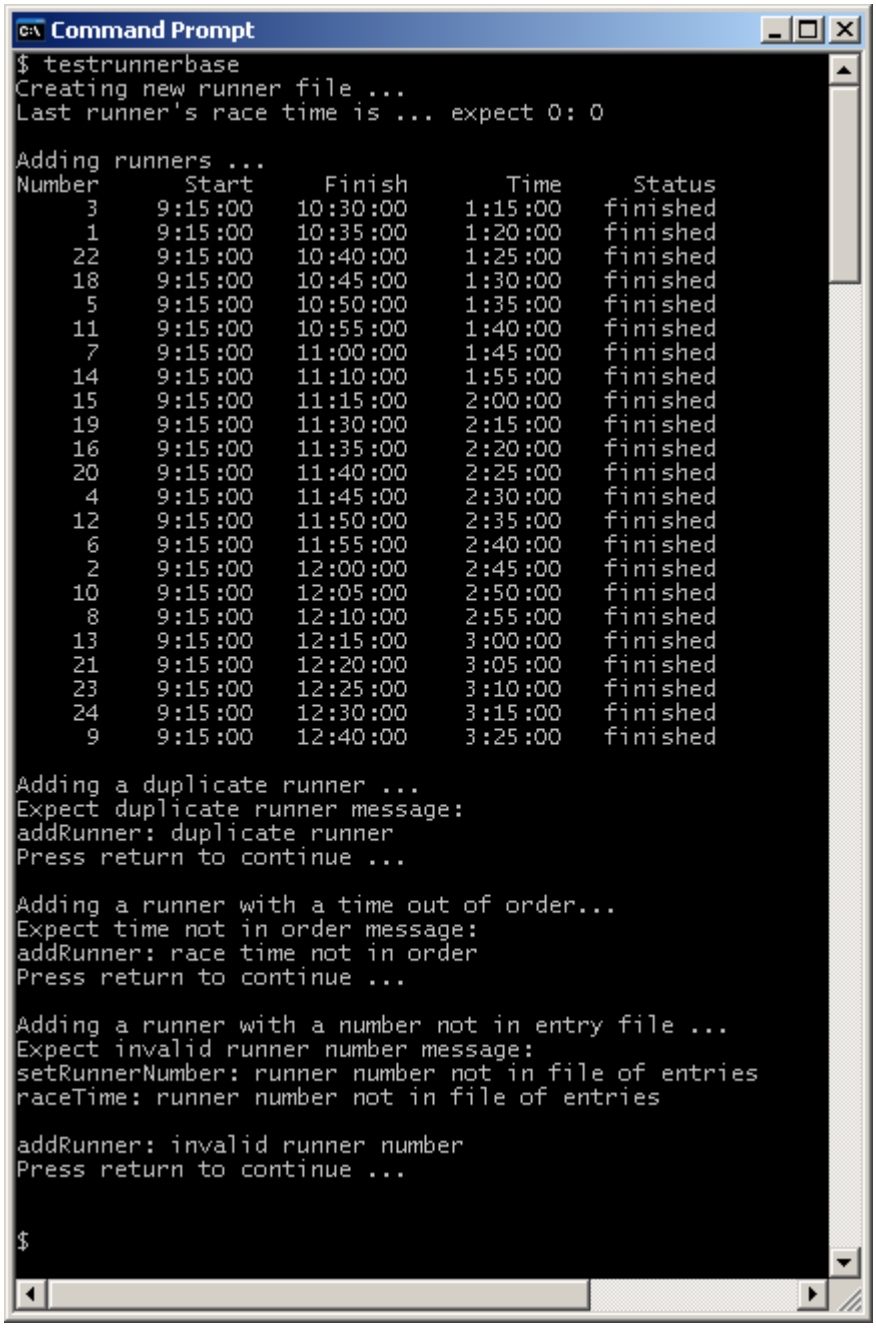
runnerbase maintains a file of runners, ensuring that they are held in race finish order.

```
/* runnerbase.c: maintains file of runners */
/* ver 1.0  20Jan2011 */

#include <stdio.h>
#include <stdlib.h>
#include "utility.h"
#include "entry.h"
#include "entrybase.h"
#include "runner.h"
```

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

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

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

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

/* NOTE: the same file handling functions appear in entryBase.
  Could there be a general file handling module?
*/

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

  file = fopen(fileName, "wb");
  if (file == NULL)
    runnerFileError("newRunnerfile: cannot open file");
  fclose(file);
  return 0;
}
```

```
/* duplicateRunner: returns 1 (true) if a runner in the file
   has the given number, 0 otherwise */
int duplicateRunner(Runner r, char *fileName)
{
  FILE *file;
  RUNNER runner;

  if (!runnerFileExists(fileName))
    runnerFileError("duplicateRunner: runner file does not exist");
  file = fopen(fileName, "rb");
  if (file == NULL)
    runnerFileError("duplicateRunner: cannot open runner file");
  fread(&runner, sizeof(RUNNER), 1, file);
  while (!feof(file)) {
    if (equalRunners(r, &runner)) {
      fclose(file);
      return 1;
    }
    fread(&runner, sizeof(RUNNER), 1, file);
  }
  fclose(file);
  return 0;
}

/* lastRaceTime: retrieves the last runner's race time
   from the runners file */
int lastRaceTime(char *fileName)
{
  FILE *file;
  RUNNER runner;
  int i = 0;

  if (!runnerFileExists(fileName))
    runnerFileError("lastRaceTime: file does not exist");
  file = fopen(fileName, "rb");
  if (file == NULL)
    runnerFileError("lastRaceTime: cannot open file");
  fread(&runner, sizeof(RUNNER), 1, file);
  while (!feof(file)) {
    i++;
    fread(&runner, sizeof(RUNNER), 1, file);
  }
  fclose(file);
  if (i == 0)
    return 0;
  return runnerRaceTime(&runner);
}

/* addRunner: appends the given runner to the file */
int addRunner(Runner runner, char *runnerFName, char *entryFName)
{
  FILE *file;
  RUNNER r;
  int result;
```

```
  if (!containsEntryNumber(runnerNumber(runner), entryFName))
    return runnerFileWarning("addRunner: invalid runner number");
  if (duplicateRunner(runner, runnerFName))
    return runnerFileWarning("addRunner: duplicate runner");
  if (lastRaceTime(runnerFName) > runnerRaceTime(runner))
    return runnerFileWarning("addRunner: race time not in order");
  r = *copyRunner(runner);
  file = fopen(runnerFName, "ab");
  if (file == NULL)
    runnerFileError("addRunner: cannot open file");
  result = fwrite(&r, sizeof(RUNNER), 1, file);
  if (result != 1)
    runnerFileError("addRunner: cannot add runner to file");
  fclose(file);
  return 0;
}

/* TO DO: create a queries file to hold runners that cannot
   be stored in the runners file (because of invalid numbers,
   times, etc.
*/

/* printRunnerFile: displays contents of runner file */
int printRunnerFile(char *fileName)
{
  FILE *file;
  RUNNER runner;

  if (!runnerFileExists(fileName))
    runnerFileError("printRunnerFile: file does not exist");
  file = fopen(fileName, "rb");
  if (file == NULL)
    runnerFileError("printRunnerFile: cannot open file");
  printRunnerHeadings();
  fread(&runner, sizeof(RUNNER), 1, file);
  while (!feof(file)) {
    printRunner(&runner);
    fread(&runner, sizeof(RUNNER), 1, file);
  }
  fclose(file);
  return 0;
}
```


Command Prompt
$ testrunnerbase
Creating new runner file ...
Last runner's race time is ... expect 0: 0
Adding runners ...
Number Start Finish Time Status
3 9:15:00 10:30:00 1:15:00 finished
1 9:15:00 10:35:00 1:20:00 finished
22 9:15:00 10:40:00 1:25:00 finished
18 9:15:00 10:45:00 1:30:00 finished
5 9:15:00 10:50:00 1:35:00 finished
11 9:15:00 10:55:00 1:40:00 finished
7 9:15:00 11:00:00 1:45:00 finished
14 9:15:00 11:10:00 1:55:00 finished
15 9:15:00 11:15:00 2:00:00 finished
19 9:15:00 11:30:00 2:15:00 finished
16 9:15:00 11:35:00 2:20:00 finished
20 9:15:00 11:40:00 2:25:00 finished
4 9:15:00 11:45:00 2:30:00 finished
12 9:15:00 11:50:00 2:35:00 finished
6 9:15:00 11:55:00 2:40:00 finished
2 9:15:00 12:00:00 2:45:00 finished
10 9:15:00 12:05:00 2:50:00 finished
8 9:15:00 12:10:00 2:55:00 finished
13 9:15:00 12:15:00 3:00:00 finished
21 9:15:00 12:20:00 3:05:00 finished
23 9:15:00 12:25:00 3:10:00 finished
24 9:15:00 12:30:00 3:15:00 finished
9 9:15:00 12:40:00 3:25:00 finished
Adding a duplicate runner ...
Expect duplicate runner message:
addRunner: duplicate runner
Press return to continue ...
Adding a runner with a time out of order...
Expect time not in order message:
addRunner: race time not in order
Press return to continue ...
Adding a runner with a number not in entry file ...
Expect invalid runner number message:
setRunnerNumber: runner number not in file of entries
raceTime: runner number not in file of entries
addRunner: invalid runner number
Press return to continue ...
$