

# CS23710 - Runners and Riders

Tom Leaman (thl5)

December 14, 2012

## 1 Design

### 1.1 Program loop & main

The main function (in `main.c`) is a very simple function which starts by reading in the data from the user supplied filenames and then executes a while loop which displays the menu, prompts for input and then calls helper functions based on what the user requires from the program.

### 1.2 Nodes & tracks

The nodes and tracks are designed as very simple data structures which hold the data as it comes out of the files. The tracks are created in duplicate, one from start node to end node and one in reverse, this simplifies the code which finds a track from two nodes and also allows for alternative safe completion times in the future (e.g. going up a hill may take longer than going down a hill).

### 1.3 Courses

The courses are similarly quite simple data structures that also hold a pointer to a vector of nodes and a pointer to a vector of tracks which comprise the course. These are used extensively when updating entrant's times and locations.

### 1.4 Entrant

The entrant is one of the most complex data structures in the program. It holds data on the id, name and course of the entrant. It also contains a pointer to the last timed/medical checkpoint node and time, the last assumed node and time and the current track and time. These are used to keep track of the entrant's location around the course as they pass checkpoints. They are also used to update which track the entrant is assumed to be on.

## 1.5 Event

The event contains the data read in from file (title, date and start time). Once the other data has been read in, it also contains a pointer to a vector of entrants and a pointer to a vector of nodes. The nodes are required for finding entrants which have gone off-track.

## 1.6 Vector

I decided to implement an array-backed vector to store the data being passed to the program. As the code does not know the size of the file before reading, I felt that a dynamically resizable data structure would be preferable and a vector seemed like the simplest option. I implemented all of the functionality that I felt would be necessary (including originally a remove function which proved to not be needed) and tested it with `vectortest.c`.

The vector uses a doubling strategy when it becomes full which means that it will only resize every  $2^n$  inserts. I think the benefit of readable, understandable (I hope) code out-weighs the extra memory allocated at the end of the vector.

I did contemplate implementing a generic linked list structure as well but I actually don't believe it to be necessary; the only time data is being shuffled around is when the vector resizes or when it is being sorted. The sorting algorithm used is bubble sort which means that during any one operation, only two sections of memory are being swapped. This (as far as I am concerned) eliminates the benefit of using a linked list (the linked list would also have more memory overhead per item due to each item needing a pointer to the next node).

## 2 Compilation

### 2.1 Main Mission

```
"/opt/csw/bin/gmake" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS=
.clean-conf
gmake[1]: Entering directory '/ceri/homes1/t/thl5/cs237-handin'
rm -f -r build/Debug
rm -f dist/Debug/NewGnu-Solaris-Sparc/cs237-handin
gmake[1]: Leaving directory '/ceri/homes1/t/thl5/cs237-handin'
```

CLEAN SUCCESSFUL (total time: 668ms)

```
"/opt/csw/bin/gmake" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS=
.build-conf
gmake[1]: Entering directory '/ceri/homes1/t/thl5/cs237-handin'
"/opt/csw/bin/gmake" -f nbproject/Makefile-Debug.mk
dist/Debug/NewGnu-Solaris-Sparc/cs237-handin
gmake[2]: Entering directory '/ceri/homes1/t/thl5/cs237-handin'
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/vector.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/vector.o.d -o
build/Debug/NewGnu-Solaris-Sparc/vector.o vector.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/track.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/track.o.d -o
build/Debug/NewGnu-Solaris-Sparc/track.o track.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/util.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/util.o.d -o
build/Debug/NewGnu-Solaris-Sparc/util.o util.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/course.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/course.o.d -o
build/Debug/NewGnu-Solaris-Sparc/course.o course.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/vectortest.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/vectortest.o.d -o
build/Debug/NewGnu-Solaris-Sparc/vectortest.o vectortest.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/main.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/main.o.d -o
build/Debug/NewGnu-Solaris-Sparc/main.o main.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/node.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/node.o.d -o
build/Debug/NewGnu-Solaris-Sparc/node.o node.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
```

```

rm -f build/Debug/NewGnu-Solaris-Sparc/event.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/event.o.d -o
build/Debug/NewGnu-Solaris-Sparc/event.o event.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/entrant.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/entrant.o.d -o
build/Debug/NewGnu-Solaris-Sparc/entrant.o entrant.c
mkdir -p dist/Debug/NewGnu-Solaris-Sparc
gcc -o dist/Debug/NewGnu-Solaris-Sparc/cs237-handin
build/Debug/NewGnu-Solaris-Sparc/vector.o
build/Debug/NewGnu-Solaris-Sparc/track.o
build/Debug/NewGnu-Solaris-Sparc/util.o
build/Debug/NewGnu-Solaris-Sparc/course.o
build/Debug/NewGnu-Solaris-Sparc/vectortest.o
build/Debug/NewGnu-Solaris-Sparc/main.o
build/Debug/NewGnu-Solaris-Sparc/node.o
build/Debug/NewGnu-Solaris-Sparc/event.o
build/Debug/NewGnu-Solaris-Sparc/entrant.o
gmake[2]: Leaving directory '/ceri/homes1/t/thl5/cs237-handin'
gmake[1]: Leaving directory '/ceri/homes1/t/thl5/cs237-handin'

```

BUILD SUCCESSFUL (total time: 10s)

## 2.2 Extended Mission

```

"/opt/csw/bin/gmake" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS=
.clean-conf
gmake[1]: Entering directory '/ceri/homes1/t/thl5/cs237-handin'
rm -f -r build/Debug
rm -f dist/Debug/NewGnu-Solaris-Sparc/cs237-handin
gmake[1]: Leaving directory '/ceri/homes1/t/thl5/cs237-handin'

```

CLEAN SUCCESSFUL (total time: 668ms)

```

"/opt/csw/bin/gmake" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS=
.build-conf
gmake[1]: Entering directory '/ceri/homes1/t/thl5/cs237-handin'
"/opt/csw/bin/gmake" -f nbproject/Makefile-Debug.mk
dist/Debug/NewGnu-Solaris-Sparc/cs237-handin
gmake[2]: Entering directory '/ceri/homes1/t/thl5/cs237-handin'
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/vector.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/vector.o.d -o
build/Debug/NewGnu-Solaris-Sparc/vector.o vector.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/track.o.d
gcc -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/track.o.d -o
build/Debug/NewGnu-Solaris-Sparc/track.o track.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc

```

```

rm -f build/Debug/NewGnu-Solaris-Sparc/util.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/util.o.d -o
build/Debug/NewGnu-Solaris-Sparc/util.o util.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/course.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/course.o.d -o
build/Debug/NewGnu-Solaris-Sparc/course.o course.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/vectortest.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/vectortest.o.d -o
build/Debug/NewGnu-Solaris-Sparc/vectortest.o vectortest.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/main.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/main.o.d -o
build/Debug/NewGnu-Solaris-Sparc/main.o main.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/node.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/node.o.d -o
build/Debug/NewGnu-Solaris-Sparc/node.o node.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/event.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/event.o.d -o
build/Debug/NewGnu-Solaris-Sparc/event.o event.c
mkdir -p build/Debug/NewGnu-Solaris-Sparc
rm -f build/Debug/NewGnu-Solaris-Sparc/entrant.o.d
gcc      -c -g -Wall -std=c89 -MMD -MP -MF
build/Debug/NewGnu-Solaris-Sparc/entrant.o.d -o
build/Debug/NewGnu-Solaris-Sparc/entrant.o entrant.c
mkdir -p dist/Debug/NewGnu-Solaris-Sparc
gcc      -o dist/Debug/NewGnu-Solaris-Sparc/cs237-handin
build/Debug/NewGnu-Solaris-Sparc/vector.o
build/Debug/NewGnu-Solaris-Sparc/track.o
build/Debug/NewGnu-Solaris-Sparc/util.o
build/Debug/NewGnu-Solaris-Sparc/course.o
build/Debug/NewGnu-Solaris-Sparc/vectortest.o
build/Debug/NewGnu-Solaris-Sparc/main.o
build/Debug/NewGnu-Solaris-Sparc/node.o
build/Debug/NewGnu-Solaris-Sparc/event.o
build/Debug/NewGnu-Solaris-Sparc/entrant.o
gmake[2]: Leaving directory '/ceri/homes1/t/thl5/cs237-handin'
gmake[1]: Leaving directory '/ceri/homes1/t/thl5/cs237-handin'

```

BUILD SUCCESSFUL (total time: 10s)

## 3 Output

### 3.1 Main Mission

Please enter name file: main\_data/name.txt  
Please enter nodes file: main\_data/nodes.txt  
Please enter tracks file: main\_data/tracks.txt  
Please enter courses file: main\_data/courses.txt  
Please enter entrants file: main\_data/entrants.txt

Endurance Horse Race - Beginners Event  
26th June 2012  
7:30

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

07:30 >> 6  
Enter checkpoint file: main\_data/cp\_times\_1.txt

Running:

1: Donald Duck  
Course: D Track: 11 Run time: 84 mins  
2: Mickey Mouse  
Course: D Track: 11 Run time: 79 mins  
3: Jemima Julieta Mouse  
Course: E Track: 16 Run time: 75 mins  
4: Minnie Duck  
Course: F Track: 14 Run time: 71 mins  
5: Minnie Mouse  
Course: E Track: 15 Run time: 67 mins  
6: Minnie Mouse Junior  
Course: E Track: 9 Run time: 63 mins  
7: Deputy Doug  
Course: D Track: 5 Run time: 58 mins  
8: Deputy Duck  
Course: D Track: 4 Run time: 53 mins  
9: Bewick Swan  
Course: F Track: 12 Run time: 49 mins  
10: Black Swan  
Course: F Track: 12 Run time: 44 mins  
11: Albert Einstein  
Course: E Track: 11 Run time: 40 mins  
12: Albert Mouse  
Course: D Track: 3 Run time: 36 mins  
13: Donald Duck Senior

Course: E Track: 2 Run time: 32 mins  
14: Egbert Einstein  
Course: F Track: 2 Run time: 28 mins

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

08:54 >> 2  
0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

08:54 >> 3  
14

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

08:54 >> 4  
0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

08:54 >> 1  
Enter entrant id: 3

3: Jemima Julieta Mouse  
Running course: E  
Started at: 07:39  
Estimated location: Track 16  
Last checkpoint: Node 9 at 08:20 (34 mins ago)  
Run time: 75 mins

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

08:54 >> 6

Enter checkpoint file: main\_data/cp\_times\_2.txt

Finished:

9: Bewick Swan  
Course: F Total time: 111 mins  
14: Egbert Einstein  
Course: F Total time: 114 mins  
2: Mickey Mouse  
Course: D Total time: 119 mins  
10: Black Swan  
Course: F Total time: 119 mins  
1: Donald Duck  
Course: D Total time: 121 mins  
3: Jemima Julieta Mouse  
Course: E Total time: 123 mins  
4: Minnie Duck  
Course: F Total time: 123 mins  
12: Albert Mouse  
Course: D Total time: 125 mins  
7: Deputy Doug  
Course: D Total time: 126 mins  
6: Minnie Mouse Junior  
Course: E Total time: 127 mins  
8: Deputy Duck  
Course: D Total time: 130 mins  
13: Donald Duck Senior  
Course: E Total time: 131 mins  
11: Albert Einstein  
Course: E Total time: 132 mins  
5: Minnie Mouse  
Course: E Total time: 138 mins

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually



6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

10:33 >> 2

0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

10:33 >> 3

0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

10:33 >> 4

14

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

10:33 >> 1

Enter entrant id: 4

4: Minnie Duck

Running course:	F
Started at:	07:43
Finished at:	09:46
Total time:	123 mins

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started

3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. Supply checkpoint times manually
6. Supply checkpoint times from a file
7. Display results list
8. Exit the program

10:33 >> 8

## 3.2 Extended Mission

Please enter name file: extended\_data/name.txt  
Please enter nodes file: extended\_data/nodes.txt  
Please enter tracks file: extended\_data/tracks.txt  
Please enter courses file: extended\_data/courses.txt  
Please enter entrants file: extended\_data/entrants.txt

Endurance Horse Race - The Main Event  
27th June 2012  
7:30

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

07:30 >> 8  
Enter checkpoint file: extended\_data/cp\_times\_1.txt

Running:

1: Ace Abbey  
Course: E Track: 1 Run time: 118 mins  
3: Ace Fudge  
Course: A Track: 18 Run time: 112 mins  
4: Amber Abbey  
Course: C Track: 13 Run time: 111 mins  
5: Amber Fudge  
Course: E Track: 13 Run time: 108 mins  
6: April Abbey  
Course: D Track: 2 Run time: 105 mins  
7: April Fudge  
Course: B Track: 17 Run time: 102 mins  
8: Ash Abbey  
Course: F Track: 13 Run time: 99 mins  
9: Ash Fudge  
Course: D Track: 2 Run time: 96 mins  
10: Asti Abbey  
Course: A Track: 15 Run time: 93 mins  
11: Asti Fudge  
Course: A Track: 15 Run time: 90 mins  
12: Autumn Abbey  
Course: C Track: 15 Run time: 87 mins  
13: Autumn Fudge  
Course: B Track: 15 Run time: 84 mins  
14: Barfields Marco Abbey  
Course: A Track: 8 Run time: 81 mins

16: Barfields Marco Fudge  
 Course: F Track: 13 Run time: 78 mins  
 17: Basil Abbey  
 Course: B Track: 8 Run time: 75 mins  
 18: Basil Fudge  
 Course: A Track: 7 Run time: 72 mins  
 19: Beatrice Abbey  
 Course: C Track: 7 Run time: 68 mins  
 20: Beatrice Fudge  
 Course: A Track: 5 Run time: 65 mins  
 22: Beau Abbey  
 Course: D Track: 5 Run time: 62 mins  
 23: Beau Fudge  
 Course: C Track: 5 Run time: 59 mins  
 24: Bella Abbey  
 Course: B Track: 4 Run time: 56 mins  
 26: Bella Fudge  
 Course: F Track: 12 Run time: 53 mins  
 27: Black Jack Abbey  
 Course: F Track: 12 Run time: 50 mins  
 28: Black Jack Fudge  
 Course: A Track: 4 Run time: 47 mins  
 30: Blue Abbey  
 Course: B Track: 4 Run time: 43 mins  
 31: Blue Fudge  
 Course: B Track: 3 Run time: 40 mins  
 32: Bobby Abbey  
 Course: A Track: 4 Run time: 37 mins  
 34: Bobby Fudge  
 Course: E Track: 11 Run time: 34 mins  
 35: Bubbles Abbey  
 Course: C Track: 2 Run time: 31 mins  
 36: Bubbles Fudge  
 Course: D Track: 2 Run time: 28 mins  
 38: Captain Abbey  
 Course: A Track: 2 Run time: 25 mins  
 39: Captain Fudge  
 Course: B Track: 2 Run time: 22 mins  
 40: Chalkie Abbey  
 Course: D Track: 1 Run time: 19 mins  
 41: Chalkie Fudge  
 Course: F Track: 1 Run time: 16 mins  
 42: Copper Abbey  
 Course: E Track: 1 Run time: 13 mins  
 44: Copper Fudge  
 Course: B Track: 1 Run time: 10 mins  
 45: Diamond Abbey  
 Course: C Track: 1 Run time: 7 mins  
 46: Diamond Fudge  
 Course: B Track: 1 Run time: 3 mins

Waiting to start:

47: Dinky Abbey  
 Course: E  
 48: Dinky Fudge  
 Course: F  
 49: Ebony Abbey  
 Course: B  
 50: Ebony Fudge

Course: C  
51: Ginger Abbey  
Course: C  
52: Ginger Fudge  
Course: F  
53: Goldie Abbey  
Course: A  
55: Goldie Fudge  
Course: E  
56: Honey Abbey  
Course: F  
57: Honey Fudge  
Course: C  
58: Izzy Abbey  
Course: A  
59: Izzy Fudge  
Course: A  
60: Jasmine Abbey  
Course: A  
61: Jasmine Fudge  
Course: F  
62: Lady Abbey  
Course: D  
64: Lady Fudge  
Course: B  
65: Lady Tara Abbey  
Course: C  
66: Lady Tara Fudge  
Course: B  
67: Lemon Abbey  
Course: B  
68: Lemon Fudge  
Course: E  
69: Lord Abbey  
Course: F  
70: Lord Fudge  
Course: E  
71: Lucky Abbey  
Course: A  
74: Lucky Fudge  
Course: E  
76: Lord Abbey  
Course: D  
77: Lord Fudge  
Course: B  
78: Maddy Abbey  
Course: F  
79: Maddy Fudge  
Course: A  
80: Magic Abbey  
Course: D  
81: Magic Fudge  
Course: D  
83: Major Abbey  
Course: A  
85: Major Fudge  
Course: A  
86: Mattie Abbey  
Course: B

87: Mattie Fudge  
Course: A  
89: Prince Abbey  
Course: B  
90: Prince Fudge  
Course: A  
91: Princess Abbey  
Course: B  
92: Princess Fudge  
Course: B  
93: Rosie Abbey  
Course: D  
94: Rosie Fudge  
Course: B  
95: Ruby Abbey  
Course: F  
97: Ruby Fudge  
Course: C  
98: Sapphire Abbey  
Course: C  
100: Sapphire Fudge  
Course: F  
101: Scarlet Abbey  
Course: C  
102: Scarlet Fudge  
Course: F  
103: sienna Abbey  
Course: D  
106: sienna Fudge  
Course: B  
107: Silver Abbey  
Course: F  
108: Silver Fudge  
Course: A  
109: Smokey Abbey  
Course: A  
110: Smokey Fudge  
Course: D  
111: Snowy Abbey  
Course: E  
113: Snowy Fudge  
Course: C  
114: sonic Abbey  
Course: A  
115: sonic Fudge  
Course: D  
117: Summer Abbey  
Course: A  
118: Summer Fudge  
Course: E  
121: Tango Abbey  
Course: B  
122: Tango Fudge  
Course: A  
123: Topaz Abbey  
Course: B  
124: Topaz Fudge  
Course: F  
126: Zizou Abbey

Course: D  
127: Zizou Fudge  
Course: F

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

09:28 >> 1  
Enter entrant id: 3

3: Ace Fudge  
Running course: A  
Started at: 07:33  
Estimated location: Track 18  
Last checkpoint: Node 14 at 09:25 (3 mins ago)  
Run time: 112 mins

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

09:28 >> 2  
64

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

09:28 >> 3  
38

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

09:28 >> 4  
0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

09:28 >> 5  
No entrants disqualified for safety reasons

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

09:28 >> 6  
No entrants disqualified for incorrect route

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list



10. Exit the program

09:28 >> 8

Enter checkpoint file: extended\_data/cp\_times\_2.txt

Finished:

26: Bella Fudge  
Course: F Total time: 109 mins  
16: Barfields Marco Fudge  
Course: F Total time: 115 mins  
8: Ash Abbey  
Course: F Total time: 116 mins  
9: Ash Fudge  
Course: D Total time: 118 mins  
6: April Abbey  
Course: D Total time: 123 mins  
1: Ace Abbey  
Course: E Total time: 124 mins  
5: Amber Fudge  
Course: E Total time: 131 mins  
4: Amber Abbey  
Course: C Total time: 157 mins

Running:

3: Ace Fudge  
Course: A Track: 18 Run time: 166 mins  
7: April Fudge  
Course: B Track: 1 Run time: 154 mins  
10: Asti Abbey  
Course: A Track: 22 Run time: 146 mins  
11: Asti Fudge  
Course: A Track: 22 Run time: 143 mins  
12: Autumn Abbey  
Course: C Track: 1 Run time: 143 mins  
13: Autumn Fudge  
Course: B Track: 13 Run time: 137 mins  
14: Barfields Marco Abbey  
Course: A Track: 20 Run time: 134 mins  
17: Basil Abbey  
Course: B Track: 13 Run time: 128 mins  
18: Basil Fudge  
Course: A Track: 18 Run time: 125 mins  
19: Beatrice Abbey  
Course: C Track: 13 Run time: 124 mins  
20: Beatrice Fudge  
Course: A Track: 18 Run time: 118 mins  
22: Beau Abbey  
Course: D Track: 1 Run time: 118 mins  
24: Bella Abbey  
Course: B Track: 17 Run time: 110 mins  
27: Black Jack Abbey  
Course: F Track: 1 Run time: 106 mins  
28: Black Jack Fudge  
Course: A Track: 17 Run time: 102 mins  
30: Blue Abbey  
Course: B Track: 17 Run time: 98 mins  
31: Blue Fudge  
Course: B Track: 15 Run time: 96 mins  
32: Bobby Abbey

Course: A Track: 15 Run time: 93 mins  
 34: Bobby Fudge  
 Course: E Track: 13 Run time: 90 mins  
 35: Bubbles Abbey  
 Course: C Track: 15 Run time: 87 mins  
 38: Captain Abbey  
 Course: A Track: 8 Run time: 81 mins  
 39: Captain Fudge  
 Course: B Track: 8 Run time: 78 mins  
 40: Chalkie Abbey  
 Course: D Track: 6 Run time: 75 mins  
 41: Chalkie Fudge  
 Course: F Track: 14 Run time: 72 mins  
 42: Copper Abbey  
 Course: E Track: 15 Run time: 69 mins  
 44: Copper Fudge  
 Course: B Track: 7 Run time: 66 mins  
 45: Diamond Abbey  
 Course: C Track: 7 Run time: 63 mins  
 46: Diamond Fudge  
 Course: B Track: 3 Run time: 59 mins  
 47: Dinky Abbey  
 Course: E Track: 9 Run time: 56 mins  
 48: Dinky Fudge  
 Course: F Track: 12 Run time: 53 mins  
 49: Ebony Abbey  
 Course: B Track: 4 Run time: 50 mins  
 50: Ebony Fudge  
 Course: C Track: 4 Run time: 47 mins  
 51: Ginger Abbey  
 Course: C Track: 4 Run time: 44 mins  
 52: Ginger Fudge  
 Course: F Track: 11 Run time: 41 mins  
 53: Goldie Abbey  
 Course: A Track: 4 Run time: 38 mins  
 55: Goldie Fudge  
 Course: E Track: 11 Run time: 35 mins  
 56: Honey Abbey  
 Course: F Track: 11 Run time: 32 mins  
 57: Honey Fudge  
 Course: C Track: 2 Run time: 28 mins  
 58: Izzy Abbey  
 Course: A Track: 2 Run time: 24 mins  
 59: Izzy Fudge  
 Course: A Track: 2 Run time: 21 mins  
 60: Jasmine Abbey  
 Course: A Track: 1 Run time: 18 mins  
 61: Jasmine Fudge  
 Course: F Track: 1 Run time: 15 mins  
 62: Lady Abbey  
 Course: D Track: 1 Run time: 12 mins  
 64: Lady Fudge  
 Course: B Track: 1 Run time: 8 mins  
 65: Lady Tara Abbey  
 Course: C Track: 1 Run time: 5 mins  
 66: Lady Tara Fudge  
 Course: B Track: 1 Run time: 1 mins

Disqualified:

23: Beau Fudge  
Course: C Disqualified for incorrect route  
36: Bubbles Fudge  
Course: D Disqualified for incorrect route

Waiting to start:

67: Lemon Abbey  
Course: B  
68: Lemon Fudge  
Course: E  
69: Lord Abbey  
Course: F  
70: Lord Fudge  
Course: E  
71: Lucky Abbey  
Course: A  
74: Lucky Fudge  
Course: E  
76: Lord Abbey  
Course: D  
77: Lord Fudge  
Course: B  
78: Maddy Abbey  
Course: F  
79: Maddy Fudge  
Course: A  
80: Magic Abbey  
Course: D  
81: Magic Fudge  
Course: D  
83: Major Abbey  
Course: A  
85: Major Fudge  
Course: A  
86: Mattie Abbey  
Course: B  
87: Mattie Fudge  
Course: A  
89: Prince Abbey  
Course: B  
90: Prince Fudge  
Course: A  
91: Princess Abbey  
Course: B  
92: Princess Fudge  
Course: B  
93: Rosie Abbey  
Course: D  
94: Rosie Fudge  
Course: B  
95: Ruby Abbey  
Course: F  
97: Ruby Fudge  
Course: C  
98: Sapphire Abbey  
Course: C  
100: Sapphire Fudge  
Course: F  
101: Scarlet Abbey

Course: C  
 102: Scarlet Fudge  
 Course: F  
 103: sienna Abbey  
 Course: D  
 106: sienna Fudge  
 Course: B  
 107: Silver Abbey  
 Course: F  
 108: Silver Fudge  
 Course: A  
 109: Smokey Abbey  
 Course: A  
 110: Smokey Fudge  
 Course: D  
 111: Snowy Abbey  
 Course: E  
 113: Snowy Fudge  
 Course: C  
 114: sonic Abbey  
 Course: A  
 115: sonic Fudge  
 Course: D  
 117: Summer Abbey  
 Course: A  
 118: Summer Fudge  
 Course: E  
 121: Tango Abbey  
 Course: B  
 122: Tango Fudge  
 Course: A  
 123: Topaz Abbey  
 Course: B  
 124: Topaz Fudge  
 Course: F  
 126: Zizou Abbey  
 Course: D  
 127: Zizou Fudge  
 Course: F

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

10:24 >> 8

Enter checkpoint file: extended\_data/cp\_times\_3.txt

Finished:

26: Bella Fudge  
 Course: F Total time: 109 mins

27: Black Jack Abbey  
 Course: F Total time: 109 mins  
 48: Dinky Fudge  
 Course: F Total time: 114 mins  
 16: Barfields Marco Fudge  
 Course: F Total time: 115 mins  
 8: Ash Abbey  
 Course: F Total time: 116 mins  
 9: Ash Fudge  
 Course: D Total time: 118 mins  
 22: Beau Abbey  
 Course: D Total time: 122 mins  
 6: April Abbey  
 Course: D Total time: 123 mins  
 34: Bobby Fudge  
 Course: E Total time: 123 mins  
 40: Chalkie Abbey  
 Course: D Total time: 123 mins  
 1: Ace Abbey  
 Course: E Total time: 124 mins  
 42: Copper Abbey  
 Course: E Total time: 125 mins  
 47: Dinky Abbey  
 Course: E Total time: 130 mins  
 5: Amber Fudge  
 Course: E Total time: 131 mins  
 19: Beatrice Abbey  
 Course: C Total time: 147 mins  
 12: Autumn Abbey  
 Course: C Total time: 150 mins  
 35: Bubbles Abbey  
 Course: C Total time: 152 mins  
 4: Amber Abbey  
 Course: C Total time: 157 mins  
 30: Blue Abbey  
 Course: B Total time: 163 mins  
 31: Blue Fudge  
 Course: B Total time: 164 mins  
 7: April Fudge  
 Course: B Total time: 166 mins  
 17: Basil Abbey  
 Course: B Total time: 169 mins  
 13: Autumn Fudge  
 Course: B Total time: 173 mins  
 24: Bella Abbey  
 Course: B Total time: 174 mins  
 3: Ace Fudge  
 Course: A Total time: 232 mins

Running:

10: Asti Abbey  
 Course: A Track: 1 Run time: 218 mins  
 11: Asti Fudge  
 Course: A Track: 1 Run time: 215 mins  
 14: Barfields Marco Abbey  
 Course: A Track: 13 Run time: 207 mins  
 18: Basil Fudge  
 Course: A Track: 13 Run time: 198 mins  
 20: Beatrice Fudge

Course: A Track: 13 Run time: 190 mins  
 28: Black Jack Fudge  
 Course: A Track: 20 Run time: 176 mins  
 32: Bobby Abbey  
 Course: A Track: 17 Run time: 162 mins  
 38: Captain Abbey  
 Course: A Track: 22 Run time: 153 mins  
 39: Captain Fudge  
 Course: B Track: 13 Run time: 151 mins  
 44: Copper Fudge  
 Course: B Track: 14 Run time: 139 mins  
 45: Diamond Abbey  
 Course: C Track: 1 Run time: 138 mins  
 49: Ebony Abbey  
 Course: B Track: 17 Run time: 122 mins  
 50: Ebony Fudge  
 Course: C Track: 13 Run time: 122 mins  
 51: Ginger Abbey  
 Course: C Track: 13 Run time: 119 mins  
 52: Ginger Fudge  
 Course: F Track: 1 Run time: 116 mins  
 53: Goldie Abbey  
 Course: A Track: 18 Run time: 111 mins  
 55: Goldie Fudge  
 Course: E Track: 13 Run time: 110 mins  
 56: Honey Abbey  
 Course: F Track: 1 Run time: 107 mins  
 57: Honey Fudge  
 Course: C Track: 16 Run time: 103 mins  
 58: Izzy Abbey  
 Course: A Track: 15 Run time: 99 mins  
 60: Jasmine Abbey  
 Course: A Track: 15 Run time: 93 mins  
 61: Jasmine Fudge  
 Course: F Track: 13 Run time: 90 mins  
 62: Lady Abbey  
 Course: D Track: 11 Run time: 87 mins  
 64: Lady Fudge  
 Course: B Track: 8 Run time: 83 mins  
 65: Lady Tara Abbey  
 Course: C Track: 8 Run time: 80 mins  
 66: Lady Tara Fudge  
 Course: B Track: 8 Run time: 76 mins  
 67: Lemon Abbey  
 Course: B Track: 8 Run time: 73 mins  
 68: Lemon Fudge  
 Course: E Track: 15 Run time: 70 mins  
 69: Lord Abbey  
 Course: F Track: 13 Run time: 67 mins  
 70: Lord Fudge  
 Course: E Track: 9 Run time: 64 mins  
 71: Lucky Abbey  
 Course: A Track: 5 Run time: 60 mins  
 74: Lucky Fudge  
 Course: E Track: 9 Run time: 57 mins  
 76: Lord Abbey  
 Course: D Track: 5 Run time: 54 mins  
 77: Lord Fudge  
 Course: B Track: 4 Run time: 50 mins

78: Maddy Abbey  
 Course: F Track: 12 Run time: 46 mins  
 79: Maddy Fudge  
 Course: A Track: 4 Run time: 43 mins  
 80: Magic Abbey  
 Course: D Track: 3 Run time: 39 mins  
 81: Magic Fudge  
 Course: D Track: 3 Run time: 35 mins  
 83: Major Abbey  
 Course: A Track: 3 Run time: 32 mins  
 85: Major Fudge  
 Course: A Track: 2 Run time: 28 mins  
 86: Mattie Abbey  
 Course: B Track: 2 Run time: 25 mins  
 87: Mattie Fudge  
 Course: A Track: 2 Run time: 22 mins  
 89: Prince Abbey  
 Course: B Track: 1 Run time: 18 mins  
 90: Prince Fudge  
 Course: A Track: 1 Run time: 15 mins  
 91: Princess Abbey  
 Course: B Track: 1 Run time: 11 mins  
 92: Princess Fudge  
 Course: B Track: 1 Run time: 8 mins  
 93: Rosie Abbey  
 Course: D Track: 1 Run time: 5 mins  
 94: Rosie Fudge  
 Course: B Track: 1 Run time: 2 mins

Disqualified:

41: Chalkie Fudge  
 Course: F Disqualified for incorrect route  
 46: Diamond Fudge  
 Course: B Disqualified for incorrect route  
 23: Beau Fudge  
 Course: C Disqualified for incorrect route  
 59: Izzy Fudge  
 Course: A Disqualified for incorrect route  
 36: Bubbles Fudge  
 Course: D Disqualified for incorrect route

Waiting to start:

95: Ruby Abbey  
 Course: F  
 97: Ruby Fudge  
 Course: C  
 98: Sapphire Abbey  
 Course: C  
 100: Sapphire Fudge  
 Course: F  
 101: Scarlet Abbey  
 Course: C  
 102: Scarlet Fudge  
 Course: F  
 103: sienna Abbey  
 Course: D  
 106: sienna Fudge  
 Course: B  
 107: Silver Abbey

Course: F  
 108: Silver Fudge  
 Course: A  
 109: Smokey Abbey  
 Course: A  
 110: Smokey Fudge  
 Course: D  
 111: Snowy Abbey  
 Course: E  
 113: Snowy Fudge  
 Course: C  
 114: sonic Abbey  
 Course: A  
 115: sonic Fudge  
 Course: D  
 117: Summer Abbey  
 Course: A  
 118: Summer Fudge  
 Course: E  
 121: Tango Abbey  
 Course: B  
 122: Tango Fudge  
 Course: A  
 123: Topaz Abbey  
 Course: B  
 124: Topaz Fudge  
 Course: F  
 126: Zizou Abbey  
 Course: D  
 127: Zizou Fudge  
 Course: F

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

11:39 >> 8

Enter checkpoint file: extended\_data/cp\_times\_4.txt

Finished:

26: Bella Fudge  
 Course: F Total time: 109 mins  
 27: Black Jack Abbey  
 Course: F Total time: 109 mins  
 48: Dinky Fudge  
 Course: F Total time: 114 mins  
 56: Honey Abbey  
 Course: F Total time: 114 mins  
 69: Lord Abbey  
 Course: F Total time: 114 mins



16: Barfields Marco Fudge  
 Course: F Total time: 115 mins  
 61: Jasmine Fudge  
 Course: F Total time: 115 mins  
 8: Ash Abbey  
 Course: F Total time: 116 mins  
 9: Ash Fudge  
 Course: D Total time: 118 mins  
 52: Ginger Fudge  
 Course: F Total time: 118 mins  
 22: Beau Abbey  
 Course: D Total time: 122 mins  
 6: April Abbey  
 Course: D Total time: 123 mins  
 34: Bobby Fudge  
 Course: E Total time: 123 mins  
 40: Chalkie Abbey  
 Course: D Total time: 123 mins  
 76: Lord Abbey  
 Course: D Total time: 123 mins  
 1: Ace Abbey  
 Course: E Total time: 124 mins  
 42: Copper Abbey  
 Course: E Total time: 125 mins  
 47: Dinky Abbey  
 Course: E Total time: 130 mins  
 5: Amber Fudge  
 Course: E Total time: 131 mins  
 55: Goldie Fudge  
 Course: E Total time: 132 mins  
 74: Lucky Fudge  
 Course: E Total time: 132 mins  
 70: Lord Fudge  
 Course: E Total time: 134 mins  
 19: Beatrice Abbey  
 Course: C Total time: 147 mins  
 45: Diamond Abbey  
 Course: C Total time: 149 mins  
 65: Lady Tara Abbey  
 Course: C Total time: 149 mins  
 12: Autumn Abbey  
 Course: C Total time: 150 mins  
 35: Bubbles Abbey  
 Course: C Total time: 152 mins  
 51: Ginger Abbey  
 Course: C Total time: 152 mins  
 50: Ebony Fudge  
 Course: C Total time: 155 mins  
 57: Honey Fudge  
 Course: C Total time: 156 mins  
 4: Amber Abbey  
 Course: C Total time: 157 mins  
 30: Blue Abbey  
 Course: B Total time: 163 mins  
 31: Blue Fudge  
 Course: B Total time: 164 mins  
 7: April Fudge  
 Course: B Total time: 166 mins  
 17: Basil Abbey

Course: B Total time: 169 mins  
39: Captain Fudge  
Course: B Total time: 171 mins  
13: Autumn Fudge  
Course: B Total time: 173 mins  
24: Bella Abbey  
Course: B Total time: 174 mins  
49: Ebony Abbey  
Course: B Total time: 184 mins  
10: Asti Abbey  
Course: A Total time: 229 mins  
14: Barfields Marco Abbey  
Course: A Total time: 229 mins  
18: Basil Fudge  
Course: A Total time: 230 mins  
20: Beatrice Fudge  
Course: A Total time: 230 mins  
11: Asti Fudge  
Course: A Total time: 231 mins  
3: Ace Fudge  
Course: A Total time: 232 mins  
32: Bobby Abbey  
Course: A Total time: 232 mins

Running:

38: Captain Abbey  
Course: A Track: 1 Run time: 225 mins  
53: Goldie Abbey  
Course: A Track: 18 Run time: 187 mins  
58: Izzy Abbey  
Course: A Track: 17 Run time: 169 mins  
62: Lady Abbey  
Course: D Track: 1 Run time: 163 mins  
60: Jasmine Abbey  
Course: A Track: 17 Run time: 162 mins  
64: Lady Fudge  
Course: B Track: 13 Run time: 157 mins  
66: Lady Tara Fudge  
Course: B Track: 13 Run time: 149 mins  
67: Lemon Abbey  
Course: B Track: 13 Run time: 145 mins  
71: Lucky Abbey  
Course: A Track: 21 Run time: 133 mins  
77: Lord Fudge  
Course: B Track: 17 Run time: 123 mins  
79: Maddy Fudge  
Course: A Track: 18 Run time: 116 mins  
80: Magic Abbey  
Course: D Track: 1 Run time: 115 mins  
81: Magic Fudge  
Course: D Track: 1 Run time: 111 mins  
83: Major Abbey  
Course: A Track: 17 Run time: 108 mins  
85: Major Fudge  
Course: A Track: 17 Run time: 104 mins  
86: Mattie Abbey  
Course: B Track: 17 Run time: 101 mins  
87: Mattie Fudge  
Course: A Track: 15 Run time: 98 mins

89: Prince Abbey  
 Course: B Track: 15 Run time: 94 mins  
 90: Prince Fudge  
 Course: A Track: 15 Run time: 91 mins  
 91: Princess Abbey  
 Course: B Track: 8 Run time: 87 mins  
 92: Princess Fudge  
 Course: B Track: 8 Run time: 84 mins  
 93: Rosie Abbey  
 Course: D Track: 11 Run time: 81 mins  
 94: Rosie Fudge  
 Course: B Track: 8 Run time: 78 mins  
 95: Ruby Abbey  
 Course: F Track: 13 Run time: 75 mins  
 97: Ruby Fudge  
 Course: C Track: 7 Run time: 72 mins  
 98: Sapphire Abbey  
 Course: C Track: 8 Run time: 69 mins  
 100: Sapphire Fudge  
 Course: F Track: 12 Run time: 66 mins  
 101: Scarlet Abbey  
 Course: C Track: 5 Run time: 63 mins  
 102: Scarlet Fudge  
 Course: F Track: 12 Run time: 60 mins  
 103: sienna Abbey  
 Course: D Track: 5 Run time: 56 mins  
 106: sienna Fudge  
 Course: B Track: 5 Run time: 53 mins  
 107: Silver Abbey  
 Course: F Track: 12 Run time: 50 mins  
 108: Silver Fudge  
 Course: A Track: 4 Run time: 47 mins  
 109: Smokey Abbey  
 Course: A Track: 4 Run time: 44 mins  
 110: Smokey Fudge  
 Course: D Track: 4 Run time: 41 mins  
 111: Snowy Abbey  
 Course: E Track: 11 Run time: 38 mins  
 113: Snowy Fudge  
 Course: C Track: 3 Run time: 35 mins  
 114: sonic Abbey  
 Course: A Track: 3 Run time: 32 mins  
 115: sonic Fudge  
 Course: D Track: 2 Run time: 29 mins  
 117: Summer Abbey  
 Course: A Track: 2 Run time: 25 mins  
 118: Summer Fudge  
 Course: E Track: 2 Run time: 22 mins  
 121: Tango Abbey  
 Course: B Track: 1 Run time: 19 mins  
 122: Tango Fudge  
 Course: A Track: 1 Run time: 16 mins  
 123: Topaz Abbey  
 Course: B Track: 1 Run time: 13 mins  
 124: Topaz Fudge  
 Course: F Track: 1 Run time: 10 mins  
 126: Zizou Abbey  
 Course: D Track: 1 Run time: 6 mins  
 127: Zizou Fudge

Course: F Track: 1 Run time: 3 mins

Disqualified:

28: Black Jack Fudge

Course: A Disqualified for incorrect route

44: Copper Fudge

Course: B Disqualified for incorrect route

68: Lemon Fudge

Course: E Disqualified for incorrect route

78: Maddy Abbey

Course: F Disqualified for incorrect route

41: Chalkie Fudge

Course: F Disqualified for incorrect route

46: Diamond Fudge

Course: B Disqualified for incorrect route

23: Beau Fudge

Course: C Disqualified for incorrect route

59: Izzy Fudge

Course: A Disqualified for incorrect route

36: Bubbles Fudge

Course: D Disqualified for incorrect route

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

12:55 >> 2

0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

12:55 >> 3

47

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course

4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

12:55 >> 8

Enter checkpoint file: extended\_data/cp\_times\_5.txt

Finished:

26: Bella Fudge  
Course: F Total time: 109 mins  
27: Black Jack Abbey  
Course: F Total time: 109 mins  
124: Topaz Fudge  
Course: F Total time: 113 mins  
48: Dinky Fudge  
Course: F Total time: 114 mins  
56: Honey Abbey  
Course: F Total time: 114 mins  
69: Lord Abbey  
Course: F Total time: 114 mins  
16: Barfields Marco Fudge  
Course: F Total time: 115 mins  
61: Jasmine Fudge  
Course: F Total time: 115 mins  
127: Zizou Fudge  
Course: F Total time: 115 mins  
8: Ash Abbey  
Course: F Total time: 116 mins  
102: Scarlet Fudge  
Course: F Total time: 116 mins  
107: Silver Abbey  
Course: F Total time: 116 mins  
100: Sapphire Fudge  
Course: F Total time: 117 mins  
9: Ash Fudge  
Course: D Total time: 118 mins  
52: Ginger Fudge  
Course: F Total time: 118 mins  
93: Rosie Abbey  
Course: D Total time: 119 mins  
95: Ruby Abbey  
Course: F Total time: 119 mins  
22: Beau Abbey  
Course: D Total time: 122 mins  
81: Magic Fudge  
Course: D Total time: 122 mins  
103: sienna Abbey  
Course: D Total time: 122 mins  
6: April Abbey  
Course: D Total time: 123 mins  
34: Bobby Fudge  
Course: E Total time: 123 mins  
40: Chalkie Abbey  
Course: D Total time: 123 mins  
76: Lord Abbey

Course: D Total time: 123 mins  
 110: Smokey Fudge  
 Course: D Total time: 123 mins  
 115: sonic Fudge  
 Course: D Total time: 123 mins  
 126: Zizou Abbey  
 Course: D Total time: 123 mins  
 1: Ace Abbey  
 Course: E Total time: 124 mins  
 80: Magic Abbey  
 Course: D Total time: 124 mins  
 42: Copper Abbey  
 Course: E Total time: 125 mins  
 47: Dinky Abbey  
 Course: E Total time: 130 mins  
 118: Summer Fudge  
 Course: E Total time: 130 mins  
 5: Amber Fudge  
 Course: E Total time: 131 mins  
 55: Goldie Fudge  
 Course: E Total time: 132 mins  
 74: Lucky Fudge  
 Course: E Total time: 132 mins  
 70: Lord Fudge  
 Course: E Total time: 134 mins  
 19: Beatrice Abbey  
 Course: C Total time: 147 mins  
 113: Snowy Fudge  
 Course: C Total time: 148 mins  
 45: Diamond Abbey  
 Course: C Total time: 149 mins  
 65: Lady Tara Abbey  
 Course: C Total time: 149 mins  
 12: Autumn Abbey  
 Course: C Total time: 150 mins  
 98: Sapphire Abbey  
 Course: C Total time: 150 mins  
 35: Bubbles Abbey  
 Course: C Total time: 152 mins  
 51: Ginger Abbey  
 Course: C Total time: 152 mins  
 50: Ebony Fudge  
 Course: C Total time: 155 mins  
 101: Scarlet Abbey  
 Course: C Total time: 155 mins  
 57: Honey Fudge  
 Course: C Total time: 156 mins  
 97: Ruby Fudge  
 Course: C Total time: 156 mins  
 4: Amber Abbey  
 Course: C Total time: 157 mins  
 30: Blue Abbey  
 Course: B Total time: 163 mins  
 31: Blue Fudge  
 Course: B Total time: 164 mins  
 94: Rosie Fudge  
 Course: B Total time: 165 mins  
 7: April Fudge  
 Course: B Total time: 166 mins

17: Basil Abbey  
Course: B Total time: 169 mins  
66: Lady Tara Fudge  
Course: B Total time: 170 mins  
39: Captain Fudge  
Course: B Total time: 171 mins  
86: Mattie Abbey  
Course: B Total time: 172 mins  
13: Autumn Fudge  
Course: B Total time: 173 mins  
24: Bella Abbey  
Course: B Total time: 174 mins  
123: Topaz Abbey  
Course: B Total time: 174 mins  
77: Lord Fudge  
Course: B Total time: 176 mins  
121: Tango Abbey  
Course: B Total time: 177 mins  
64: Lady Fudge  
Course: B Total time: 179 mins  
89: Prince Abbey  
Course: B Total time: 179 mins  
91: Princess Abbey  
Course: B Total time: 181 mins  
67: Lemon Abbey  
Course: B Total time: 182 mins  
92: Princess Fudge  
Course: B Total time: 182 mins  
49: Ebony Abbey  
Course: B Total time: 184 mins  
83: Major Abbey  
Course: A Total time: 223 mins  
108: Silver Fudge  
Course: A Total time: 223 mins  
114: sonic Abbey  
Course: A Total time: 227 mins  
10: Asti Abbey  
Course: A Total time: 229 mins  
14: Barfields Marco Abbey  
Course: A Total time: 229 mins  
18: Basil Fudge  
Course: A Total time: 230 mins  
20: Beatrice Fudge  
Course: A Total time: 230 mins  
60: Jasmine Abbey  
Course: A Total time: 230 mins  
11: Asti Fudge  
Course: A Total time: 231 mins  
3: Ace Fudge  
Course: A Total time: 232 mins  
32: Bobby Abbey  
Course: A Total time: 232 mins  
38: Captain Abbey  
Course: A Total time: 232 mins  
87: Mattie Fudge  
Course: A Total time: 232 mins  
58: Izzy Abbey  
Course: A Total time: 233 mins  
79: Maddy Fudge

Course: A Total time: 234 mins  
109: Smokey Abbey  
Course: A Total time: 235 mins  
117: Summer Abbey  
Course: A Total time: 238 mins  
90: Prince Fudge  
Course: A Total time: 240 mins  
122: Tango Fudge  
Course: A Total time: 242 mins

Disqualified:

71: Lucky Abbey  
Course: A Disqualified for incorrect route  
85: Major Fudge  
Course: A Disqualified for incorrect route  
53: Goldie Abbey  
Course: A Disqualified for incorrect route  
28: Black Jack Fudge  
Course: A Disqualified for incorrect route  
106: sienna Fudge  
Course: B Disqualified for incorrect route  
62: Lady Abbey  
Course: D Disqualified for incorrect route  
44: Copper Fudge  
Course: B Disqualified for incorrect route  
68: Lemon Fudge  
Course: E Disqualified for incorrect route  
78: Maddy Abbey  
Course: F Disqualified for incorrect route  
41: Chalkie Fudge  
Course: F Disqualified for incorrect route  
46: Diamond Fudge  
Course: B Disqualified for incorrect route  
23: Beau Fudge  
Course: C Disqualified for incorrect route  
111: Snowy Abbey  
Course: E Disqualified for incorrect route  
59: Izzy Fudge  
Course: A Disqualified for incorrect route  
36: Bubbles Fudge  
Course: D Disqualified for incorrect route

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 1  
Enter entrant id: 71

71: Lucky Abbey



Running course: A  
Started at: 10:39  
Disqualified at: Node 9 at 13:56

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 2  
0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 3  
0

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 4  
87

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety

6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 5

No entrants disqualified for safety reasons

Please select from the following options:

1. Locate a entrant
2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 6

71: Lucky Abbey

Course: A Last node: 9

85: Major Fudge

Course: A Last node: 7

53: Goldie Abbey

Course: A Last node: 17

28: Black Jack Fudge

Course: A Last node: 13

106: sienna Fudge

Course: B Last node: 13

62: Lady Abbey

Course: D Last node: 17

44: Copper Fudge

Course: B Last node: 5

68: Lemon Fudge

Course: E Last node: 14

78: Maddy Abbey

Course: F Last node: 17

41: Chalkie Fudge

Course: F Last node: 7

46: Diamond Fudge

Course: B Last node: 13

23: Beau Fudge

Course: C Last node: 9

111: Snowy Abbey

Course: E Last node: 4

59: Izzy Fudge

Course: A Last node: 7

36: Bubbles Fudge

Course: D Last node: 17

Total disqualified for incorrect route: 15

Please select from the following options:

1. Locate a entrant

2. Show how many entrants have not yet started
3. Show how many entrants are currently on the course
4. Show how many entrants have finished
5. List entrants excluded for safety
6. List entrants excluded for incorrect route
7. Supply checkpoint times manually
8. Supply checkpoint times from a file
9. Display results list
10. Exit the program

16:48 >> 10

## 4 Source

### 4.1 Main Mission

All the source for the main mission is identical except for main.c

```
/*
 * File:    main.c
 * Author:  thl5
 *
 * Created on 14 December 2012, 10:04
 */

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

#include "vector.h"
#include "util.h"
#include "data.h"

/*
 * returns a valid filename
 */
char* get_filename(char* prompt) {
    char* filename;

    printf("%s", prompt);
    filename = readline();
    while (!valid_filename(filename)) {
        printf("Could not open %s\n", filename);
        printf("%s", prompt);
        free(filename);
        filename = readline();
    }
    return filename;
}

/*
 * reads in data from all relevant files
 * returns a pointer to an event object
 */
Event* read_data() {
    char* filename;
    Event* event;
    Vector* nodes;
    Vector* tracks;
    Vector* courses;
    Vector* entrants;

    filename = get_filename("Please enter name file: ");
    event = read_event(filename);

    filename = get_filename("Please enter nodes file: ");
```

```

nodes = read_nodes(filename);

filename = get_filename("Please enter tracks file: ");
tracks = read_tracks(filename, nodes);

filename = get_filename("Please enter courses file: ");
courses = read_courses(filename, nodes, tracks);

filename = get_filename("Please enter entrants file: ");
entrants = read_entrants(filename, courses);

event->nodes = nodes;
event->entrants = entrants;
return event;
}

/*
 * print out the name and details of the event
 */
void display_event_header(Event* event) {
    printf("\n\n");
    printf("\t%s\n", event->title);
    printf("\t%s\n", event->date);
    printf("\t%d:%d\n", event->start->hours, event->start->minutes);
    printf("\n");
}

/*
 * display the menu and grab an int from the user
 * to specify what action to take
 */
int display_menu(Event* event) {
    char* line;
    char* token;
    int input;
    printf("\n");
    printf("Please select from the following options:\n");
    printf("\n");
    printf("\t 1. Locate a entrant\n");
    printf("\t 2. Show how many entrants have not yet started\n");
    printf("\t 3. Show how many entrants are currently on the course\n");
    printf("\t 4. Show how many entrants have finished\n");
    /* printf("\t 5. List entrants excluded for safety\n");
    printf("\t 6. List entrants excluded for incorrect route\n"); */
    printf("\t 5. Supply checkpoint times manually\n");
    printf("\t 6. Supply checkpoint times from a file\n");
    printf("\t 7. Display results list\n");
    printf("\t 8. Exit the program\n");
    printf("\n");

    printf("%02d:%02d >> ", event->time->hours, event->time->minutes);
    line = readline();
    token = strtok(line, "\n");
    input = atoi(token);
    free(line);

    return input;
}

```

```

/*
 * locate a particular entrant from id
 */
void locate_entrant(Event* event) {
    Entrant* entrant;
    int entrant_id;
    char* line;

    printf("Enter entrant id: ");
    line = readline();
    strtok(line, "\n");
    entrant_id = atoi(line);

    entrant = entrant_from_id(event->entrants, entrant_id);
    entrant_stats(entrant, event->time);
    free(line);
}

/*
 * returns how many entrants have a particular status
 */
int count_by_status(Event* event, entrant_status status) {
    Entrant* entrant;
    int ret_val = 0;
    int i = 0;

    for (i = 0; i < Vector_size(event->entrants); i++) {
        Vector_get(event->entrants, i, &entrant);
        if (entrant->status == status) ret_val++;
    }

    return ret_val;
}

/*
 * list entrants excluded for safety reasons
 */
void list_excluded_safety(Event* event) {
    Entrant* entrant;
    int i = 0;

    /* make sure there's at least one */
    if (count_by_status(event, DISQUAL_SAFETY) == 0) {
        printf("\tNo entrants disqualified for safety reasons\n");
    } else {
        for (i = 0; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status == DISQUAL_SAFETY) {
                printf("\t%2d: %-50s\n", entrant->id, entrant->name);
                printf("\t\tCourse: %c Last node: %2d\n", entrant->course->id,
                    entrant->last_cp_node->id);
            }
        }
        /* print total */
        printf("\n\tTotal disqualified for safety reasons: %d\n",
            count_by_status(event, DISQUAL_SAFETY));
    }
}

```

```

/*
 * list entrants excluded for getting lost
 */
void list_excluded_incorrect(Event* event) {
    Entrant* entrant;
    int i = 0;

    /* make sure there's at least one */
    if (count_by_status(event, DISQUAL_INCORR) == 0) {
        printf("\tNo entrants disqualified for incorrect route\n");
    } else {
        for (i = 0; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status == DISQUAL_INCORR) {
                printf("\t\t%2d: %-50s\n", entrant->id, entrant->name);
                printf("\t\t\tCourse: %c Last node: %2d\n", entrant->course->id,
                    entrant->last_cp_node->id);
            }
        }
        /* print total */
        printf("\n\tTotal disqualified for incorrect route: %d\n",
            count_by_status(event, DISQUAL_INCORR));
    }
}

/*
 * display all the entrants based on status/duration
 */
void display_results(Event* event) {
    Entrant* entrant;
    int i = 0;

    entrants_sort(event);
    /* since the entrants are now sorted, we know that all entrants with a particular
     * status will be grouped together */

    /* display finished */
    if (count_by_status(event, FINISHED) > 0) {
        printf("\n\tFinished:\n");
        for (; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status != FINISHED) break; /* if entrant hasn't finished, skip to
                                                    the next block of entrants */
            printf("\t\t\t%3d: %-50s\n", entrant->id, entrant->name);
            printf("\t\t\t\tCourse: %c Total time: %3d mins\n", entrant->course->id, entrant->duration);
        }
    }
    /* display started and stopped */
    if (count_by_status(event, STARTED) + count_by_status(event, STOPPED) > 0) {
        printf("\n\tRunning:\n");
        for (; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status != STARTED && entrant->status != STOPPED) break;

            printf("\t\t\t%3d: %-50s\n", entrant->id, entrant->name);
            printf("\t\t\t\tCourse: %c Track: %2d Run time: %3d mins\n", entrant->course->id,
                entrant->curr_track->id, entrant->duration);
        }
    }
}

```

```

/* display disqualified */
if (count_by_status(event, DISQUAL_SAFETY) + count_by_status(event, DISQUAL_INCORR) > 0) {
    printf("\n\tDisqualified:\n");
    for (; i < Vector_size(event->entrants); i++) {
        Vector_get(event->entrants, i, &entrant);
        if (entrant->status != DISQUAL_SAFETY && entrant->status != DISQUAL_INCORR) break;

        printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
        printf("\t\t\tCourse: %c ", entrant->course->id);
        if (entrant->status == DISQUAL_SAFETY) printf("Disqualified for safety\n");
        else printf("Disqualified for incorrect route\n");
    }
}
/* display waiting to start */
if (count_by_status(event, NOT_STARTED) > 0) {
    printf("\n\tWaiting to start:\n");
    for (; i < Vector_size(event->entrants); i++) {
        Vector_get(event->entrants, i, &entrant);

        printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
        printf("\t\t\tCourse: %c\n", entrant->course->id);
    }
}
}

/*
 * update an entrant manually
 */
void update_manual(Event* event) {
    char* line;
    char type;
    int node_id;
    int entrant_id;
    Time* time;

    /* type */
    printf("Enter update type (T/I/A/D/E): ");
    line = readline();
    type = line[0];
    while (type != 't' && type != 'T' &&
           type != 'i' && type != 'I' &&
           type != 'a' && type != 'A' &&
           type != 'd' && type != 'D' &&
           type != 'e' && type != 'E') {
        printf("Invalid type. Please enter one of T/I/A/D/E: ");
        free(line);
        line = readline();
        type = line[0];
    }
    free(line);

    /* node id */
    printf("Enter node id: ");
    line = readline();
    node_id = atoi(line);
    free(line);

    /* entrant id */
    printf("Enter entrant id: ");

```



```

line = readline();
entrant_id = atoi(line);
free(line);

/* time */
printf("Enter time (hh:mm): ");
line = readline();
time = str_to_time(line);
free(line);

update_time(event, time, entrant_id); /* entrant_id refers to an entrant that will
                                     NOT be updated by this call */
entrant_update_location(event, type, entrant_id, node_id); /* they get updated here */

/* print out the entrant's stats */
entrant_stats(entrant_from_id(event->entrants, entrant_id), event->time);
free(time);
}

/*
 * update entrants from a file
 */
void update_file(Event* event) {
    char* filename = get_filename("Enter checkpoint file: ");
    Vector* lines = read_file(filename);
    char* line;
    char* token;
    char type;
    int node_id;
    int entrant_id;
    Time* time;
    int i = 0;

    for (i = 0; i < Vector_size(lines); i++) {
        Vector_get(lines, i, &line);

        /* type */
        token = strtok(line, " ");
        type = token[0];

        /* node id */
        token = strtok(NULL, " ");
        node_id = atoi(token);

        /* entrant id */
        token = strtok(NULL, " ");
        entrant_id = atoi(token);

        /* time */
        token = strtok(NULL, "\n");
        time = str_to_time(token);

        update_time(event, time, entrant_id);
        entrant_update_location(event, type, entrant_id, node_id);
    }

    display_results(event);
    Vector_dispose(lines);
}

```

```

/*
 * the main method (including program loop)
 */
int main(int argc, char** argv) {
    Event* event = read_data();
    int running = 1;
    int input;

    display_event_header(event);
    while (running) {
        input = display_menu(event);
        switch (input) {
            case 1:
                locate_entrant(event);
                break;
            case 2:
                printf("\t%d\n", count_by_status(event, NOT_STARTED));
                break;
            case 3:
                printf("\t%d\n", count_by_status(event, STARTED) + count_by_status(event, STOPPED));
                break;
            case 4:
                printf("\t%d\n", count_by_status(event, FINISHED));
                break;
            /* case 5:
                list_excluded_safety(event);
                break;
            case 6:
                list_excluded_incorrect(event);
                break; */
            case 5:
                update_manual(event);
                break;
            case 6:
                update_file(event);
                break;
            case 7:
                display_results(event);
                break;
            case 8:
                running = 0;
                break;
            default:
                /* invalid input, do nothing */
                break;
        }
    }

    return (EXIT_SUCCESS);
}

```

## 4.2 Extended Mission

### 4.2.1 vector.h

```
/*
 * File:    vector.h
 * Author:  thl5
 *
 * Created on 14 December 2012, 10:04
 */

#ifndef VECTOR_H
#define VECTOR_H

#ifdef __cplusplus
extern "C" {
#endif

#define MIN_CAPACITY 4

/*
 * This was designed based on principles from Stanford University's CS107 lectures
 * available on YouTube. In particular the lecture where they describe implementing
 * a generic stack in C.
 *
 * This code (and that in vector.c) is, however all my own work
 */

typedef struct Vector {
    size_t elem_size;          /* the size of the elements being stored */
    void (*dispose_fn)(void*); /* a function which knows how to dispose of the elements
                                being stored */
    int size;                  /* the logical size of the vector */
    int capacity;              /* the capacity of the vector */
    void* base;                /* a pointer to the start of the data */
} Vector;

Vector* Vector_new(int elem_size, void (*dispose_fn)(void*));
void Vector_dispose(Vector* vector);

/* add an element to the end */
void Vector_add(Vector* vector, void* elem);
/* get the value of an element at index */
void Vector_get(Vector* vector, int index, void* ret_val);
/* sorts the data in the vector according to the comparison function */
void Vector_sort(Vector* vector, int (*comp_fn)(void*, void*));
/* the size of the vector */
int Vector_size(Vector* vector);

#ifdef __cplusplus
}
#endif

#endif /* VECTOR_H */
```

#### 4.2.2 vector.c

```
#include <stdlib.h>
#include <string.h>

#include "vector.h"

/*
 * Note: we can't do pointer arithmetic on void* pointers, the compiler doesn't
 * know how big the thing it's pointing to is!
 * So we use a cast to a char* (which is one byte) to multiply the pointer
 * arithmetic by 1 and point to (hopefully) the correct location.
 */

/*
 * private functions
 */

static void Vector_grow(Vector* vector) {
    void* new_base;

    /* use a doubling strategy when the vector is full
     * this means the vector will only resize every 2^n inserts */
    vector->capacity *= 2;
    /* allocate new space */
    new_base = malloc(vector->elem_size * vector->capacity);
    /* copy the old data over */
    memcpy(new_base, vector->base, vector->elem_size * vector->size);
    /* free the old space */
    free(vector->base);
    /* update the local pointer */
    vector->base = new_base;
}

/*
 * functions declared in vector.h
 */

/* create a new vector object */
Vector* Vector_new(int elem_size, void (*dispose_fn)(void*)) {
    Vector* vector = malloc(sizeof(Vector));

    vector->elem_size = elem_size;
    vector->dispose_fn = dispose_fn;
    vector->size = 0;
    vector->capacity = MIN_CAPACITY;
    vector->base = malloc(vector->elem_size * vector->capacity);

    return vector;
}

/* dispose of the vector */
void Vector_dispose(Vector* vector) {
    int i = 0;

    if (vector) {
```

```

    if (vector->dispose_fn) {
        /* this will only be used if a disposal function has been passed in */
        for (i = 0; i < vector->size; i++) {
            vector->dispose_fn((char*)vector->base + vector->elem_size * i);
        }
    }

    if (vector->base) free(vector->base);
    free(vector);
}

/* add an element to the vector */
void Vector_add(Vector* vector, void* elem) {
    void* target;

    if (vector) {
        if (vector->size == vector->capacity) Vector_grow(vector);

        /* target will point to the next empty space in the vector */
        target = (char*) vector->base + vector->elem_size * vector->size++;
        /* copy the data in */
        memcpy(target, elem, vector->elem_size);
    }
}

/* copy the value at index into ret_val */
void Vector_get(Vector* vector, int index, void* ret_val) {
    if (vector) {
        if (index >= 0 && index < vector->size)
            memcpy(ret_val, (char*)vector->base + vector->elem_size * index, vector->elem_size);
    }
}

/* this implements a generic bubble sort */
void Vector_sort(Vector* vector, int (*comp_fn)(void*, void*)) {
    int i = 0;
    int j = 0;
    void* temp = malloc(vector->elem_size);

    if (vector) {
        for (i = 0; i < vector->size; i++) {
            for (j = 1; j < vector->size; j++) {
                /* a = j-1, b = j */
                if (comp_fn((char*)vector->base + vector->elem_size * (j-1),
                    (char*)vector->base + vector->elem_size * j) == 1) { /* if a > b */
                    /* temp = a */
                    memcpy(temp, (char*)vector->base + vector->elem_size * (j-1), vector->elem_size);
                    /* a = b */
                    memcpy((char*)vector->base + vector->elem_size * (j-1),
                        (char*)vector->base + vector->elem_size * j, vector->elem_size);
                    /* b = temp */
                    memcpy((char*)vector->base + vector->elem_size * j, temp, vector->elem_size);
                }
            }
        }
        free(temp);
    }
}

```

```

/* just return the size */
int Vector_size(Vector* vector) {
    /* I could let the client use vector->size directly, but I think it's more
     * consistent to get them to call a function */
    return vector->size;
}

```

### 4.2.3 vectortest.c

```

/*
 * File:    vectortest.c
 * Author:  thl5
 *
 * Created on 14 December 2012, 10:10
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "vector.h"
#include "util.h"

void test_add_simple(Vector* vector) {
    int i = 0;
    for (i = 1; i < 11; i++) {
        printf("Adding: %d ", i);
        Vector_add(vector, &i);
        printf(" Size: %d Capacity: %d\n", vector->size, vector->capacity);
    }
}

void test_get_simple(Vector* vector) {
    int ret_val;
    printf("Getting first elem: ");
    Vector_get(vector, 0, &ret_val);
    printf("%d Size: %d Capacity: %d\n", ret_val, vector->size, vector->capacity);
}

void test_simple() {
    Vector* vector = Vector_new(sizeof(int), NULL);
    test_add_simple(vector);
    printf("\n");
    test_get_simple(vector);
    Vector_dispose(vector);
}

void dispose_str(void* str) {
    char* foo = *(char**)str;
    if (foo) free(foo);
}

void test_add_complex(Vector* vector) {
    char* str;
    printf("Adding 'Fred': ");
}

```

```

    str = strdup("Fred");
    Vector_add(vector, &str);
    printf("Size: %d Capacity: %d\n", vector->size, vector->capacity);
    printf("Adding 'Wilma': ");
    str = strdup("Wilma");
    Vector_add(vector, &str);
    printf("Size: %d Capacity: %d\n", vector->size, vector->capacity);
    printf("Adding 'Barney': ");
    str = strdup("Barney");
    Vector_add(vector, &str);
    printf("Size: %d Capacity: %d\n", vector->size, vector->capacity);
    printf("Adding 'Pebbles': ");
    str = strdup("Pebbles");
    Vector_add(vector, &str);
    printf("Size: %d Capacity: %d\n", vector->size, vector->capacity);
    printf("Adding 'Bam-Bam': ");
    str = strdup("Bam-Bam");
    Vector_add(vector, &str);
    printf("Size: %d Capacity: %d\n", vector->size, vector->capacity);
}

void test_get_complex(Vector* vector) {
    char* str;
    printf("Getting first elem: ");
    Vector_get(vector, 0, &str);
    printf("%s Size: %d Capacity: %d\n", str, vector->size, vector->capacity);
}

void test_complex() {
    Vector* vector = Vector_new(sizeof(char**), dispose_str);
    test_add_complex(vector);
    printf("\n");
    test_get_complex(vector);
    /* no real need to test out of bounds again */
    Vector_dispose(vector);
}

/*
 *
 * Uncomment this and comment main to run
int main(int argc, char** argv) {
    printf("\n\nTesting simple\n\n");
    test_simple();
    printf("\n\nTesting complex\n\n");
    test_complex();
    return (EXIT_SUCCESS);
}
*/

```

#### 4.2.4 util.h

```

/*
 * File:    util.h
 * Author:  thl5
 *

```

```

    * Created on 14 December 2012, 10:08
    */

#ifndef UTIL_H
#define UTIL_H

#ifdef __cplusplus
extern "C" {
#endif

#define MAX_LINE_LENGTH 80

typedef struct Time {
    int hours;
    int minutes;
} Time;

/* 'borrowed' functions */
char* strdup(const char* str);      /* returns a pointer to a newly malloc'd string */
char* readline();                  /* reads a line from stdin */

int valid_filename(char* filename); /* checks if a filename is valid */
Vector* read_file(char* filename); /* reads a whole file into a vector */
Time* str_to_time(char* str);       /* produces a time from a string (hh:mm) */
Time* timecpy(Time* time);          /* deep copy a time struct */
int time_to_duration(Time* time);   /* turn hh:mm into mm */

#ifdef __cplusplus
}
#endif

#endif /* UTIL_H */

```

#### 4.2.5 util.c

```

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

#include "vector.h"
#include "util.h"

/*
 * private functions
 */

/* this gets passed into the vectors that are used to read file data
 * it means we can call Vector_dispose once we've read the data in */
void string_dispose(void* string) {
    /* this is a bit of a funky cast, we're casting it to a char** (which it is) */
    /* and then dereferencing that to produce our char* foo */
    char* foo = *(char**) string;
    if (foo) free(foo);
}

```



```

/*
 * functions declared in util.h
 */

/*
 * This function was taken from
 * http://cboard.cprogramming.com/c-programming/
 * 95462-compiler-error-warning-implicit-declaration-function-strdup.html
 * as it's not part of the c89 standard but it is really *really* useful
 * See ref [1]
 */
char* strdup(const char* str) {
    int n = strlen(str) + 1;
    char* dup = malloc(n);
    if (dup) {
        strcpy(dup, str);
    }
    return dup;
}

/*
 * The following function was taken from
 * http://stackoverflow.com/questions/314401/how-to-read-a-line-from-the-console-in-c
 * I could have used scanf, but I think just reading in a whole line each time and
 * then parsing it makes everything easier to read in the rest of the source
 * See ref [2]
 */
char* readline() {
    char* line = malloc(100), *linep = line;
    size_t lenmax = 100, len = lenmax;
    int c;

    if(line == NULL)
        return NULL;

    for(;;) {
        c = fgetc(stdin);
        if(c == EOF)
            break;

        if(--len == 0) {
            len = lenmax;
            char * linen = realloc(linep, lenmax *= 2);

            if(linen == NULL) {
                free(linep);
                return NULL;
            }
            line = linen + (line - linep);
            linep = linen;
        }

        if((*line++ = c) == '\n')
            break;
    }
    *line = '\0';
    strtok(linep, "\n"); /* this line is mine, just to strip the newline */
    return linep;
}

```

```

}

/* check that a filename is valid (1 = true, 0 = false) */
int valid_filename(char* filename) {
    FILE* fp;
    int ret_val = 0;

    fp = fopen(filename, "r");
    if (fp) {
        ret_val = 1;
        fclose(fp);
    }

    return ret_val;
}

/* this reads a whole file into a vector */
Vector* read_file(char* filename) {
    char line[MAX_LINE_LENGTH];
    char* str;
    FILE* fp;
    Vector* lines = Vector_new(sizeof(char**), string_dispose);

    fp = fopen(filename, "r");
    while (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
        strtok(line, "\n"); /* strip newline */
        str = strdup(line); /* make a new pointer */
        Vector_add(lines, &str); /* pass it to the vector */
    }
    fclose(fp);

    return lines;
}

/* turn a string (hh:mm) into a struct Time */
Time* str_to_time(char* str) {
    Time* time = malloc(sizeof(Time));
    char* token;

    token = strtok(str, ":");
    time->hours = atoi(token);
    token = strtok(NULL, "\n");
    time->minutes = atoi(token);

    return time;
}

/* return a new copy of a struct Time */
Time* timecpy(Time* time) {
    Time* copy = malloc(sizeof(Time));
    copy->hours = time->hours;
    copy->minutes = time->minutes;
    return copy;
}

/* return a duration from a time */
int time_to_duration(Time* time) {
    return time->minutes + (time->hours * 60);
}

```

#### 4.2.6 data.h

```
/*
 * File:   data.h
 * Author: thl5
 *
 * Created on 14 December 2012, 10:14
 */

#ifndef DATA_H
#define DATA_H

#ifdef __cplusplus
extern "C" {
#endif

/*
 * event
 */

typedef struct Event {
    char* title;
    char* date;
    Time* start;
    Vector* nodes;
    Vector* entrants;
    Time* time;
} Event;

Event* read_event(char* filename);
/* will not update entrant_id, that's handled by entrant_update_location */
void update_time(Event* event, Time* time, int entrant_id);

/*
 * node
 */

typedef enum {
    CP,
    MC,
    JN
} node_type;

typedef struct Node {
    int id;
    node_type type;
} Node;

Vector* read_nodes(char* filename);
Node* node_from_id(Vector* nodes, int id);
node_type str_to_type(char* str);

/*
 * track
 */

typedef struct Track {
```

```

    int id;
    Node* start;
    Node* end;
    int safe_time;
} Track;

Vector* read_tracks(char* filename, Vector* nodes);
Track* track_from_nodes(Vector* tracks, Node* start, Node* end);

/*
 * course
 */

typedef struct Course {
    char id;
    Vector* nodes;
    Vector* tracks;
    int safe_time;
} Course;

Vector* read_courses(char* filename, Vector* nodes, Vector* tracks);
Course* course_from_id(Vector* courses, char id);
Track* next_track(Course* course, Track* current);
/* grabs the next track in the course and accounts for entrants arriving so
 * early that they confuse the track predictions from entrant_update_time */
Track* next_track_from_node(Course* course, Track* current, Node* node);

/*
 * entrant
 */

typedef enum {
    NOT_STARTED,    /* waiting to start */
    STARTED,        /* running */
    STOPPED,        /* at a medical checkpoint */
    DISQUAL_SAFETY, /* disqualified for safety */
    DISQUAL_INCORR, /* disqualified for getting lost */
    FINISHED        /* finished */
} entrant_status;

typedef struct Entrant {
    int id;
    Course* course;
    char* name;

    int duration; /* the run time */
    entrant_status status;
    Time* start_time;
    /* the last timed point */
    Node* last_cp_node;
    Time* last_cp_time;
    /* the assumed location */
    Node* last_node;
    Time* last_time;
    /* the current time and what track we think they're on */
    Time* curr_time;
    Track* curr_track;
} Entrant;

```

```

Vector* read_entrants(char* filename, Vector* courses);
Entrant* entrant_from_id(Vector* entrants, int id);
void entrant_stats(Entrant* entrant, Time* curr_time);
/* updates the entrant at a timed or medical checkpoint */
void entrant_update_location(Event* event, char type, int entrant_id, int node_id);
/* updates the running time and predicts which track they're on */
void entrant_update_time(Event* event, Entrant* entrant);
void entrants_sort(Event* event);

#ifdef __cplusplus
}
#endif

#endif /* DATA_H */

```

#### 4.2.7 node.c

```

#include <stdlib.h>
#include <string.h>

#include "vector.h"
#include "util.h"
#include "data.h"

/*
 * functions declared in data.h
 */

/* read nodes from the file */
Vector* read_nodes(char* filename) {
    Vector* lines = read_file(filename);
    Vector* nodes = Vector_new(sizeof(Node*), NULL);
    Node* node;
    char* line;
    char* token;
    int i = 0;

    for (i = 0; i < Vector_size(lines); i++) {
        Vector_get(lines, i, &line);
        node = malloc(sizeof(Node));

        /* id */
        token = strtok(line, " ");
        node->id = atoi(token);

        /* type */
        token = strtok(NULL, " ");
        node->type = str_to_type(token);

        Vector_add(nodes, &node);
    }

    Vector_dispose(lines);
    return nodes;
}

```

```

}

/* find a node from id */
Node* node_from_id(Vector* nodes, int id) {
    Node* node;
    int i = 0;

    for (i = 0; i < Vector_size(nodes); i++) {
        Vector_get(nodes, i, &node);
        if (node->id == id) return node;
    }

    return NULL;
}

/* get a node type from a string */
node_type str_to_type(char* str) {
    if (strcmp(str, "CP") == 0) return CP;
    else if (strcmp(str, "MC") == 0) return MC;
    else return JN;
}

```

#### 4.2.8 track.c

```

#include <stdlib.h>
#include <string.h>

#include "vector.h"
#include "util.h"
#include "data.h"

/*
 * functions declared in data.h
 */

/* read tracks from the file */
Vector* read_tracks(char* filename, Vector* nodes) {
    Vector* lines = read_file(filename);
    Vector* tracks = Vector_new(sizeof(Track*), NULL);
    Track* track_for;
    Track* track_back;
    Node* start_node;
    Node* end_node;
    char* line;
    char* token;
    int node_id;
    int i = 0;

    /*
     * Tracks are being added as a pair, one forward and one backward
     */

    for (i = 0; i < Vector_size(lines); i++) {
        Vector_get(lines, i, &line);
        track_for = malloc(sizeof(Track));

```

```

track_back = malloc(sizeof(Track));

/* id */
token = strtok(line, " ");
track_for->id = atoi(token);
track_back->id = atoi(token);

/* nodes */
token = strtok(NULL, " ");
node_id = atoi(token);
start_node = node_from_id(nodes, node_id);

token = strtok(NULL, " ");
node_id = atoi(token);
end_node = node_from_id(nodes, node_id);

track_for->start = start_node;
track_for->end = end_node;
track_back->start = end_node;
track_back->end = start_node;

/* safe time */
token = strtok(NULL, " ");
track_for->safe_time = atoi(token);
track_back->safe_time = atoi(token);

Vector_add(tracks, &track_for);
Vector_add(tracks, &track_back);
}

Vector_dispose(lines);
return tracks;
}

/* find a track from start node to end node */
Track* track_from_nodes(Vector* tracks, Node* start, Node* end) {
    Track* track;
    int i = 0;

    for (i = 0; i < Vector_size(tracks); i++) {
        Vector_get(tracks, i, &track);
        if (track->start == start && track->end == end) return track;
    }

    return NULL;
}

```

#### 4.2.9 course.c

```

#include <stdlib.h>
#include <string.h>

#include "vector.h"
#include "util.h"

```

```

#include "data.h"

/*
 * functions declared in data.h
 */

/* read in the courses from the file */
Vector* read_courses(char* filename, Vector* nodes, Vector* tracks) {
    Vector* lines = read_file(filename);
    Vector* courses = Vector_new(sizeof(Course*), NULL);
    Course* course;
    Node* node;
    Node* start;
    Node* end;
    Track* track;
    char* line;
    char* token;
    int num_nodes;
    int node_id;
    int i, j = 0;

    for (i = 0; i < Vector_size(lines); i++) {
        Vector_get(lines, i, &line);
        course = malloc(sizeof(Course));

        /* id */
        token = strtok(line, " ");
        course->id = token[0]; /* 1 char */

        /* nodes */
        course->nodes = Vector_new(sizeof(Node*), NULL);
        token = strtok(NULL, " ");
        num_nodes = atoi(token);

        for (j = 0; j < num_nodes; j++) {
            token = strtok(NULL, " ");
            node_id = atoi(token);

            node = node_from_id(nodes, node_id);
            Vector_add(course->nodes, &node);
        }

        /* tracks + safe_time */
        course->tracks = Vector_new(sizeof(Track*), NULL);
        course->safe_time = 0;
        for (j = 0; j < num_nodes - 1; j++) { /* num tracks = num_nodes - 1 */
            Vector_get(course->nodes, j, &start);
            Vector_get(course->nodes, j + 1, &end);

            track = track_from_nodes(tracks, start, end);
            Vector_add(course->tracks, &track);

            course->safe_time += track->safe_time;
        }

        Vector_add(courses, &course);
    }

    Vector_dispose(lines);
}

```



```

    return courses;
}

/* find a course by id */
Course* course_from_id(Vector* courses, char id) {
    Course* course;
    int i = 0;

    for (i = 0; i < Vector_size(courses); i++) {
        Vector_get(courses, i, &course);
        if (course->id == id) return course;
    }

    return NULL;
}

/* find the next track in the course */
Track* next_track(Course* course, Track* current) {
    Track* track;
    Track* next;
    int i = 0;

    /* start one beyond the start of the vector so we don't overrun */
    for (i = 1; i < Vector_size(course->tracks); i++) {
        Vector_get(course->tracks, i - 1, &track);
        if (track == current) {
            Vector_get(course->tracks, i, &next);
            return next;
        }
    }

    return NULL;
}

/* finds the next track after current starting with node */
Track* next_track_from_node(Course* course, Track* current, Node* node) {
    Track* next = next_track(course, current);
    while (next && next->start != node) /* while this isn't the track we're searching for */
        next = next_track(course, next);
    return next;
}

```

#### 4.2.10 entrant.c

```

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

#include "vector.h"
#include "util.h"
#include "data.h"

/*
 * private functions

```

```

*/

/* turn a status into a string */
char* status_to_str(entrant_status status) {
    if (status == NOT_STARTED) {
        return "Waiting to start";
    } else if (status == STARTED) {
        return "Started";
    } else if (status == STOPPED) {
        return "At medical checkpoint";
    } else if (status == DISQUAL_SAFETY) {
        return "Disqualified for safety";
    } else if (status == DISQUAL_INCORR) {
        return "Disqualified for incorrect route";
    } else {
        return "Finished";
    }
}

/* compares entrants (used by the vector to sort entrants) */
int compare_entrants(void* vp1, void* vp2) {
    /*
    * a < b   -  -1
    * a = b   -   0
    * a > b   -   1
    */
    Entrant* a = *(Entrant**)vp1;
    Entrant* b = *(Entrant**)vp2;

    /*
    * Put finished at the top
    * ascending by duration
    *
    * Then started/stopped
    * descending by duration
    *
    * The disqualified
    * descending by duration
    *
    * Then not started
    * ascending by id
    */

    if (a->status == FINISHED) {
        if (b->status == FINISHED) {
            if (a->duration < b->duration) return -1;
            else if (a->duration > b->duration) return 1;
            else return 0;
        } else return -1; /* b is not finished */

    } else if (a->status == STARTED || a->status == STOPPED) {
        if (b->status == FINISHED) return 1;
        else if (b->status == STARTED || b->status == STOPPED) {
            if (a->duration < b->duration) return 1;
            else if (a->duration > b->duration) return -1;
            else return 0;
        } else return -1; /* b is either not started or is disqualified */

    } else if (a->status == DISQUAL_SAFETY || a->status == DISQUAL_INCORR) {

```

```

    if (b->status == FINISHED || b->status == STARTED || b->status == STOPPED) return 1;
    else if (b->status == DISQUAL_SAFETY || b->status == DISQUAL_INCORR) {
        if (a->duration < b->duration) return 1;
        else if (a->duration > b->duration) return -1;
        else return 0;
    } else return -1; /* b has not even started */

} else { /* a has not started */
    if (b->status != NOT_STARTED) return 1;
    else {
        if (a->id < b->id) return -1;
        else return 1; /* ids will not be equal */
    }
}
}

/* update the last_cp_node and time (and current node and time)
 * of the entrant */
void update_nodes(Entrant* entrant, Node* node, Time* time) {
    entrant->last_cp_node = node;
    if (entrant->last_cp_time) free(entrant->last_cp_time);
    entrant->last_cp_time = timecpy(time);

    entrant->last_node = node;
    if (entrant->last_time) free(entrant->last_time);
    entrant->last_time = timecpy(time);
}

/*
 * functions declared in data.h
 */

/* read entrants from the file */
Vector* read_entrants(char* filename, Vector* courses) {
    Vector* lines = read_file(filename);
    Vector* entrants = Vector_new(sizeof(Entrant*), NULL);
    Entrant* entrant;
    char* line;
    char* token;
    int i = 0;

    for (i = 0; i < Vector_size(lines); i++) {
        Vector_get(lines, i, &line);
        entrant = malloc(sizeof(Entrant));

        /* id */
        token = strtok(line, " ");
        entrant->id = atoi(token);

        /* course */
        token = strtok(NULL, " ");
        entrant->course = course_from_id(courses, token[0]); /* 1 char */

        /* name */
        token = strtok(NULL, "\n"); /* read to end of line */
        entrant->name = strdup(token);

        /* other data
         * these all get updated as the entrant starts in entrant_update_location */

```

```

    entrant->duration = 0;
    entrant->status = NOT_STARTED;
    entrant->start_time = NULL;
    entrant->last_cp_node = NULL;
    entrant->last_cp_time = NULL;
    entrant->last_node = NULL;
    entrant->last_time = NULL;
    entrant->curr_time = NULL;
    entrant->curr_track = NULL;

    Vector_add(entrants, &entrant);
}

Vector_dispose(lines);
return entrants;
}

/* find an entrant by id */
Entrant* entrant_from_id(Vector* entrants, int id) {
    Entrant* entrant;
    int i = 0;

    for (i = 0; i < Vector_size(entrants); i++) {
        Vector_get(entrants, i, &entrant);
        if (entrant->id == id) return entrant;
    }

    return NULL;
}

/* print out the entrants stats */
void entrant_stats(Entrant* entrant, Time* curr_time) {
    /* generic data for all entrants */
    printf("\n");
    printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
    printf("\t\tRunning course:          %c\n", entrant->course->id);

    if (entrant->status == NOT_STARTED) {
        printf("\t\tWaiting to start\n");
    } else {
        printf("\t\tStarted at:              %02d:%02d\n",
            entrant->start_time->hours,
            entrant->start_time->minutes);
        /*
        * started
        */
        if (entrant->status == STARTED) {
            printf("\t\tEstimated location:      Track %d\n", entrant->curr_track->id);
            printf("\t\tLast checkpoint:          Node %d at %02d:%02d (%d mins ago)\n",
                entrant->last_cp_node->id,
                entrant->last_cp_time->hours,
                entrant->last_cp_time->minutes,
                (time_to_duration(curr_time) - time_to_duration(entrant->last_cp_time)));
            printf("\t\tRun time:                  %d mins\n", entrant->duration);
            /*
            * stopped
            */
        } else if (entrant->status == STOPPED) {
            printf("\t\tAt medical checkpoint: Node %d since %02d:%02d (%d mins ago)\n",

```

```

        entrant->last_cp_node->id, entrant->last_cp_time->hours,
        entrant->last_cp_time->minutes,
        (time_to_duration(curr_time) - time_to_duration(entrant->last_cp_time)));
    printf("\t\tRun time:                %d mins\n", entrant->duration);
/*
 * disqualified - safety
 */
} else if (entrant->status == DISQUAL_SAFETY) {
    printf("\t\tExcluded for safety at: Node %d at %02d:%02d\n",
        entrant->last_cp_node->id,
        entrant->last_cp_time->hours,
        entrant->last_cp_time->minutes);
/*
 * disqualified - bad route
 */
} else if (entrant->status == DISQUAL_INCORR) {
    printf("\t\tDisqualified at:            Node %d at %02d:%02d\n",
        entrant->last_cp_node->id,
        entrant->last_cp_time->hours,
        entrant->last_cp_time->minutes);
/*
 * finished
 */
} else if (entrant->status == FINISHED) {
    printf("\t\tFinished at:                %02d:%02d\n",
        entrant->last_cp_time->hours,
        entrant->last_cp_time->minutes);
    printf("\t\tTotal time:                %d mins\n", entrant->duration);
}
}
}

/* update the entrant's location */
void entrant_update_location(Event* event, char type, int entrant_id, int node_id) {
    Entrant* entrant = entrant_from_id(event->entrants, entrant_id);
    Node* node = node_from_id(event->nodes, node_id);

    /* This entrant will not have had entrant_update_time called on it */

    /*****
     * T type
     *****/
    if (type == 't' || type == 'T') {
        /* check if not started and init */
        if (entrant->status == NOT_STARTED) {
            entrant->status = STARTED;
            entrant->start_time = timecpy(event->time);
            entrant->curr_time = timecpy(event->time);
        }

        update_nodes(entrant, node, event->time);

        /* update duration and current time */
        entrant->duration += time_to_duration(event->time) -
            time_to_duration(entrant->curr_time);
        if (entrant->curr_time) free(entrant->curr_time);
        entrant->curr_time = timecpy(event->time);

        /* update current track */

```

```

    if (entrant->curr_track)
        entrant->curr_track =
            next_track_from_node(entrant->course, entrant->curr_track, node);
    else
        /* _might_ not be initd (on starting) so init to tracks[0] */
        Vector_get(entrant->course->tracks, 0, &entrant->curr_track);

    /* check if finished */
    if (entrant->curr_track == NULL)
        entrant->status = FINISHED;

    /*****
    * I type
    *****/
} else if (type == 'i' || type == 'I') {
    update_nodes(entrant, node, event->time);

    /* update duration and current time */
    entrant->duration += time_to_duration(event->time) -
        time_to_duration(entrant->curr_time);
    if (entrant->curr_time) free(entrant->curr_time);
    entrant->curr_time = timecpy(event->time);

    entrant->status = DISQUAL_INCORR;

    /*****
    * A type
    *****/
} else if (type == 'a' || type == 'A') {
    update_nodes(entrant, node, event->time);

    entrant->duration += time_to_duration(event->time) -
        time_to_duration(entrant->curr_time);

    entrant->status = STOPPED;

    /*****
    * D type
    *****/
} else if (type == 'd' || type == 'D') {
    update_nodes(entrant, node, event->time);

    /* set current time */
    if (entrant->curr_time) free(entrant->curr_time);
    entrant->curr_time = timecpy(event->time);

    /* update next track and status */
    entrant->curr_track =
        next_track_from_node(entrant->course, entrant->curr_track, node);
    entrant->status = STARTED;

    /*****
    * E type
    *****/
} else { /* type == 'e' */
    entrant->status = DISQUAL_SAFETY;
}
}

```

```

/* update the entrant's time and estimated location */
void entrant_update_time(Event* event, Entrant* entrant) {
    int time_since_seen; /* time since last node */
    int time_delta; /* time since this function was last called */

    if (entrant->status == STARTED) {
        /* if they've started, they'd better not have null values */

        /* calc some useful values */
        time_since_seen = time_to_duration(event->time) -
            time_to_duration(entrant->last_time);
        time_delta = time_to_duration(event->time) -
            time_to_duration(entrant->curr_time);

        /* update duration and curr_time */
        entrant->duration += time_delta;
        if (entrant->curr_time) free(entrant->curr_time);
        entrant->curr_time = timecpy(event->time);

        if (entrant->curr_track->end->type == JN && /* next junction is not timed */
            time_since_seen > entrant->curr_track->safe_time) { /* entrant has finished this track */
            /* find next track */
            entrant->curr_track = next_track(entrant->course, entrant->curr_track);
            /* update last node & time */
            entrant->last_node = entrant->curr_track->start;
            if (entrant->last_time) free(entrant->last_time);
            entrant->last_time = timecpy(event->time);
        }
    }
}

/* sort the entrants */
void entrants_sort(Event* event) {
    Vector_sort(event->entrants, compare_entrants);
}

```

#### 4.2.11 event.c

```

#include <stdlib.h>
#include <string.h>

#include "vector.h"
#include "util.h"
#include "data.h"

/*
 * functions declared in data.h
 */

/* read an event from file
 * doesn't initialise everything! */
Event* read_event(char* filename) {
    Vector* lines = read_file(filename);
    Event* event = malloc(sizeof(Event));

```

```

char* line;
char* token;

Vector_get(lines, 0, &line);
token = strtok(line, "\n");
event->title = strdup(token);

Vector_get(lines, 1, &line);
token = strtok(line, "\n");
event->date = strdup(token);

Vector_get(lines, 2, &line);
event->start = str_to_time(line);

event->time = timecpy(event->start);

Vector_dispose(lines);
return event;
}

/* update the time in event and the entrants
 * does not update the entrant identified by entrant_id,
 * this will be done by a subsequent call to entrant_update_location */
void update_time(Event* event, Time* time, int entrant_id) {
    Entrant* entrant;
    int i = 0;

    free(event->time);
    event->time = timecpy(time);

    /* now update the entrants */
    for (i = 0; i < Vector_size(event->entrants); i++) {
        Vector_get(event->entrants, i, &entrant);
        if (entrant->id != entrant_id)
            entrant_update_time(event, entrant);
    }
}

```

#### 4.2.12 main.c

```

/*
 * File:   main.c
 * Author: thl5
 *
 * Created on 14 December 2012, 10:04
 */

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

#include "vector.h"
#include "util.h"
#include "data.h"

```



```

/*
 * returns a valid filename
 */
char* get_filename(char* prompt) {
    char* filename;

    printf("%s", prompt);
    filename = readline();
    while (!valid_filename(filename)) {
        printf("Could not open %s\n", filename);
        printf("%s", prompt);
        free(filename);
        filename = readline();
    }
    return filename;
}

/*
 * reads in data from all relevant files
 * returns a pointer to an event object
 */
Event* read_data() {
    char* filename;
    Event* event;
    Vector* nodes;
    Vector* tracks;
    Vector* courses;
    Vector* entrants;

    filename = get_filename("Please enter name file: ");
    event = read_event(filename);

    filename = get_filename("Please enter nodes file: ");
    nodes = read_nodes(filename);

    filename = get_filename("Please enter tracks file: ");
    tracks = read_tracks(filename, nodes);

    filename = get_filename("Please enter courses file: ");
    courses = read_courses(filename, nodes, tracks);

    filename = get_filename("Please enter entrants file: ");
    entrants = read_entrants(filename, courses);

    event->nodes = nodes;
    event->entrants = entrants;
    return event;
}

/*
 * print out the name and details of the event
 */
void display_event_header(Event* event) {
    printf("\n\n");
    printf("\t%s\n", event->title);
    printf("\t%s\n", event->date);
    printf("\t%d:%d\n", event->start->hours, event->start->minutes);
    printf("\n");
}

```

```

/*
 * display the menu and grab an int from the user
 * to specify what action to take
 */
int display_menu(Event* event) {
    char* line;
    char* token;
    int input;
    printf("\n");
    printf("Please select from the following options:\n");
    printf("\n");
    printf("\t 1. Locate a entrant\n");
    printf("\t 2. Show how many entrants have not yet started\n");
    printf("\t 3. Show how many entrants are currently on the course\n");
    printf("\t 4. Show how many entrants have finished\n");
    printf("\t 5. List entrants excluded for safety\n");
    printf("\t 6. List entrants excluded for incorrect route\n");
    printf("\t 7. Supply checkpoint times manually\n");
    printf("\t 8. Supply checkpoint times from a file\n");
    printf("\t 9. Display results list\n");
    printf("\t10. Exit the program\n");
    printf("\n");

    printf("%02d:%02d >> ", event->time->hours, event->time->minutes);
    line = readline();
    token = strtok(line, "\n");
    input = atoi(token);
    free(line);

    return input;
}

/*
 * locate a particular entrant from id
 */
void locate_entrant(Event* event) {
    Entrant* entrant;
    int entrant_id;
    char* line;

    printf("Enter entrant id: ");
    line = readline();
    strtok(line, "\n");
    entrant_id = atoi(line);

    entrant = entrant_from_id(event->entrants, entrant_id);
    entrant_stats(entrant, event->time);
    free(line);
}

/*
 * returns how many entrants have a particular status
 */
int count_by_status(Event* event, entrant_status status) {
    Entrant* entrant;
    int ret_val = 0;
    int i = 0;

```

```

    for (i = 0; i < Vector_size(event->entrants); i++) {
        Vector_get(event->entrants, i, &entrant);
        if (entrant->status == status) ret_val++;
    }

    return ret_val;
}

/*
 * list entrants excluded for safety reasons
 */
void list_excluded_safety(Event* event) {
    Entrant* entrant;
    int i = 0;

    /* make sure there's at least one */
    if (count_by_status(event, DISQUAL_SAFETY) == 0) {
        printf("\tNo entrants disqualified for safety reasons\n");
    } else {
        for (i = 0; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status == DISQUAL_SAFETY) {
                printf("\t%2d: %-50s\n", entrant->id, entrant->name);
                printf("\t\tCourse: %c Last node: %2d\n", entrant->course->id,
                    entrant->last_cp_node->id);
            }
        }
        /* print total */
        printf("\n\tTotal disqualified for safety reasons: %d\n",
            count_by_status(event, DISQUAL_SAFETY));
    }
}

/*
 * list entrants excluded for getting lost
 */
void list_excluded_incorrect(Event* event) {
    Entrant* entrant;
    int i = 0;

    /* make sure there's at least one */
    if (count_by_status(event, DISQUAL_INCORR) == 0) {
        printf("\tNo entrants disqualified for incorrect route\n");
    } else {
        for (i = 0; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status == DISQUAL_INCORR) {
                printf("\t%2d: %-50s\n", entrant->id, entrant->name);
                printf("\t\tCourse: %c Last node: %2d\n", entrant->course->id,
                    entrant->last_cp_node->id);
            }
        }
        /* print total */
        printf("\n\tTotal disqualified for incorrect route: %d\n",
            count_by_status(event, DISQUAL_INCORR));
    }
}

/*

```

```

    * display all the entrants based on status/duration
    */
void display_results(Event* event) {
    Entrant* entrant;
    int i = 0;

    entrants_sort(event);
    /* since the entrants are now sorted, we know that all entrants with a particular
    * status will be grouped together */

    /* display finished */
    if (count_by_status(event, FINISHED) > 0) {
        printf("\n\tFinished:\n");
        for (; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status != FINISHED) break; /* if entrant hasn't finished, skip to
                                                    the next block of entrants */
            printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
            printf("\t\t\tCourse: %c Total time: %3d mins\n", entrant->course->id, entrant->duration);
        }
    }
    /* display started and stopped */
    if (count_by_status(event, STARTED) + count_by_status(event, STOPPED) > 0) {
        printf("\n\tRunning:\n");
        for (; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status != STARTED && entrant->status != STOPPED) break;

            printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
            printf("\t\t\tCourse: %c Track: %2d Run time: %3d mins\n", entrant->course->id,
                entrant->curr_track->id, entrant->duration);
        }
    }
    /* display disqualified */
    if (count_by_status(event, DISQUAL_SAFETY) + count_by_status(event, DISQUAL_INCORR) > 0) {
        printf("\n\tDisqualified:\n");
        for (; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);
            if (entrant->status != DISQUAL_SAFETY && entrant->status != DISQUAL_INCORR) break;

            printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
            printf("\t\t\tCourse: %c ", entrant->course->id);
            if (entrant->status == DISQUAL_SAFETY) printf("Disqualified for safety\n");
            else printf("Disqualified for incorrect route\n");
        }
    }
    /* display waiting to start */
    if (count_by_status(event, NOT_STARTED) > 0) {
        printf("\n\tWaiting to start:\n");
        for (; i < Vector_size(event->entrants); i++) {
            Vector_get(event->entrants, i, &entrant);

            printf("\t\t%3d: %-50s\n", entrant->id, entrant->name);
            printf("\t\t\tCourse: %c\n", entrant->course->id);
        }
    }
}
/*

```

```

    * update an entrant manually
    */
void update_manual(Event* event) {
    char* line;
    char type;
    int node_id;
    int entrant_id;
    Time* time;

    /* type */
    printf("Enter update type (T/I/A/D/E): ");
    line = readline();
    type = line[0];
    while (type != 't' && type != 'T' &&
           type != 'i' && type != 'I' &&
           type != 'a' && type != 'A' &&
           type != 'd' && type != 'D' &&
           type != 'e' && type != 'E') {
        printf("Invalid type. Please enter one of T/I/A/D/E: ");
        free(line);
        line = readline();
        type = line[0];
    }
    free(line);

    /* node id */
    printf("Enter node id: ");
    line = readline();
    node_id = atoi(line);
    free(line);

    /* entrant id */
    printf("Enter entrant id: ");
    line = readline();
    entrant_id = atoi(line);
    free(line);

    /* time */
    printf("Enter time (hh:mm): ");
    line = readline();
    time = str_to_time(line);
    free(line);

    update_time(event, time, entrant_id); /* entrant_id refers to an entrant that will
                                           NOT be updated by this call */
    entrant_update_location(event, type, entrant_id, node_id); /* they get updated here */

    /* print out the entrant's stats */
    entrant_stats(entrant_from_id(event->entrants, entrant_id), event->time);
    free(time);
}

/*
 * update entrants from a file
 */
void update_file(Event* event) {
    char* filename = get_filename("Enter checkpoint file: ");
    Vector* lines = read_file(filename);
    char* line;

```

```

char* token;
char type;
int node_id;
int entrant_id;
Time* time;
int i = 0;

for (i = 0; i < Vector_size(lines); i++) {
    Vector_get(lines, i, &line);

    /* type */
    token = strtok(line, " ");
    type = token[0];

    /* node id */
    token = strtok(NULL, " ");
    node_id = atoi(token);

    /* entrant id */
    token = strtok(NULL, " ");
    entrant_id = atoi(token);

    /* time */
    token = strtok(NULL, "\n");
    time = str_to_time(token);

    update_time(event, time, entrant_id);
    entrant_update_location(event, type, entrant_id, node_id);
}

display_results(event);
Vector_dispose(lines);
}

/*
 * the main method (including program loop)
 */
int main(int argc, char** argv) {
    Event* event = read_data();
    int running = 1;
    int input;

    display_event_header(event);
    while (running) {
        input = display_menu(event);
        switch (input) {
            case 1:
                locate_entrant(event);
                break;
            case 2:
                printf("\t%d\n", count_by_status(event, NOT_STARTED));
                break;
            case 3:
                printf("\t%d\n", count_by_status(event, STARTED) + count_by_status(event, STOPPED));
                break;
            case 4:
                printf("\t%d\n", count_by_status(event, FINISHED));
                break;
            case 5:

```

```

        list_excluded_safety(event);
        break;
case 6:
    list_excluded_incorrect(event);
    break;
case 7:
    update_manual(event);
    break;
case 8:
    update_file(event);
    break;
case 9:
    display_results(event);
    break;
case 10:
    running = 0;
    break;
default:
    /* invalid input, do nothing */
    break;
}
}

return (EXIT_SUCCESS);
}

```

## 5 References

- 1. <http://cboard.cprogramming.com/c-programming/95462-compiler-error-warning-implicit-declaration-function-strdup.html> (util.c)
- 2. <http://stackoverflow.com/questions/314401/how-to-read-a-line-from-the-console-in-c> (util.c)