Runners and Riders
A CS23710 Assignment

Ben Brooks (beb12@aber.ac.uk)

Due date: 2012-12-14 16:00

Contents

Abstract

Runners and Riders is a CS23710 assignment written in C. It aims to track positions and times of entrants in cross country events.

1 Source Code

1.1 Header Files

1.1.1 runners.h

```
/*
 * File:   runners.h
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#ifndef RUNNERS_H
#define RUNNERS_H

#ifdef __cplusplus
extern "C" {
#endif

#define FILE_LENGTH 256
#define TIME_LENGTH 6
#define NAME_LENGTH 50
#define COURSE_LENGTH 2
#define TIME_TYPE_LENGTH 2
#define NODE_TYPE_LENGTH 3
#define COURSE_NODE_MAX 20

    void menu(void);

#ifdef __cplusplus
}
#endif

#endif /* RUNNERS_H */
```

1.1.2 event_data.h

```
/*
```

```
 * File:   event_data.h
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#ifndef RUNNERS_AND_RIDERS_H
#define RUNNERS_AND_RIDERS_H

#ifdef __cplusplus
extern "C" {
#endif

    typedef struct {
        char event_name[80];
        char event_date[80];
        char event_time[6];
    } event;

    typedef struct node {
        int node_number;
        char node_type[3];
        struct node *next_node;
    } node;

    typedef struct track {
        int track_number;
        int track_start_node;
        int track_end_node;
        int track_avg_time;
        struct track *next_track;
    } track;

    typedef struct course {
        char course_identifier[2];
        int course_number_of_nodes;
        int course_nodes[20]; /* Todo: clean up array size */
        struct course *next_course;
    } course;

    void read_event_file(char *str);
    void print_event_data(void);
    void current_time(char *str);
    void read_node_file(char *str);
    void print_node_list(void);
    void read_track_file(char *str);
    void print_track_list(void);
    void read_course_file(char *str);
    void print_course_list(void);
    int get_first_node(char *str);

#ifdef __cplusplus
}
#endif

#endif /* RUNNERS_AND_RIDERS_H */
```

1.1.3 entrant_data.h

```
/*
 * File:   entrant_data.h
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#ifndef ENTRANT_DATA_H
#define ENTRANT_DATA_H

#ifdef __cplusplus
extern "C" {
#endif

    typedef struct entrant {
        int entrant_number;
        char entrant_course[COURSE_LENGTH];
        char entrant_name[NAME_LENGTH];
        int first_node;
        int last_seen;
        char start_time[TIME_LENGTH];
        char end_time[TIME_LENGTH];
        struct entrant *next_entrant;
    } entrant;

    void read_entrant_file(char *str);
    void print_entrant_list(void);
    void print_current_status(int entrant_number);
    void update_entrant_location(int entrant_number, int node, char *str);
    void print_unstarted_entrants(void);
    void print_on_course_entrants(void);
    void print_finished_entrants(void);
    void print_results_table(void);

#ifdef __cplusplus
}
#endif

#endif /* ENTRANT_DATA_H */
```

1.1.4 time_data.h

```
/*
 * File:   time_data.h
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#ifndef TIME_DATA_H
#define TIME_DATA_H

#ifdef __cplusplus
extern "C" {
#endif

    void read_time_file(char *str);
    int time_diff(char *str, char *str2);

#ifdef __cplusplus
}
#endif

#endif /* TIME_DATA_H */
```

1.2 C Source Files

1.2.1 main.c

```
/*
```

```
 * File:   main.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "runners.h"
#include "event_data.h"
#include "entrant_data.h"
#include "time_data.h"

/*#define DEBUG*/

int main(int argc, char** argv) {

char eventfile[FILE_LENGTH];
char nodefile[FILE_LENGTH];
char trackfile[FILE_LENGTH];
char coursefile[FILE_LENGTH];
char entrantfile[FILE_LENGTH];

#ifdef DEBUG
    strcpy(&eventfile, "../../data/main/name.txt");
    strcpy(&nodefile, "../../data/main/nodes.txt");
    strcpy(&trackfile, "../../data/main/tracks.txt");
    strcpy(&coursefile, "../../data/main/courses.txt");
    strcpy(&entrantfile, "../../data/main/entrants.txt");
#else
printf("Event file: ");
scanf(" %255s", eventfile);
printf("Node file: ");
scanf(" %255s", nodefile);
printf("Track file: ");
scanf(" %255s", trackfile);
printf("Course file: ");
scanf(" %255s", coursefile);
printf("Entrant file: ");
scanf(" %255s", entrantfile);
#endif
```

```
    read_event_file(eventfile);
    read_node_file(nodefile);
    read_track_file(trackfile);
    read_course_file(coursefile);
    read_entrant_file(entrantfile);

#ifdef DEBUG
    printf("--BEGIN DEBUG INFO--\n");
    print_node_list();
    printf("--------------------\n");
    print_track_list();
    printf("--------------------\n");
    print_course_list();
    printf("--------------------\n");
    print_entrant_list();
    printf("--END DEBUG INFO-----\n\n");
#endif

    menu();
return (EXIT_SUCCESS);

}

void menu() {
int runloop = 1;
while (runloop) {
int selection = 0;
        int entrant_id = 0;
        int checkpoint_id = 0;
        char time[TIME_LENGTH];
        char timefile[FILE_LENGTH];
        int diff = 0;
        printf("--------------------\n");
        print_event_data();
        printf("--------------------\n");
printf("  1. Query the current location of a competitor\n");
printf("  2. List competitors which haven't started yet\n");
printf("  3. List competitors which are out on the course\n");
printf("  4. List competitors who have finished\n");
printf("  5. Manually supply times for a competitor\n");
printf("  6. Read in a file containing times\n");
printf("  7. Print a results list\n");
printf("  0. Quit\n");

printf("Choose an option: ");
scanf("%d", &selection);

    switch (selection) {
case 1:
            printf("Enter Entrant ID: ");
            scanf("%d", &entrant_id);
            print_current_status(entrant_id);
break;
case 2:
            print_unstarted_entrants();
break;
        case 3:
            print_on_course_entrants();
            break;
        case 4:
            print_finished_entrants();
            break;
        case 5:
            printf("Enter Entrant ID: ");
            scanf("%d", &entrant_id);
            printf("Enter Checkpoint Number: ");
            scanf("%d", &checkpoint_id);
            printf("Enter Time: ");
            scanf(" %5s", time);
            update_entrant_location(entrant_id, checkpoint_id, time);
            break;
        case 6:
            printf("Enter time file: ");
            scanf(" %255s", timefile);
            read_time_file(timefile);
            break;
        case 7:
            print_results_table();
            break;
        case 0:
printf("Bye Bye!\n");
runloop = 0;
break;
default:
printf("\nInvalid Selection!\n\n");
break;
}
}
}
```

1.2.2 event.c

```
/*
 * File:   event.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "runners.h"
#include "event_data.h"

event the_event;

/*
 * Function to print event data
 */
void print_event_data(void) {
    printf("%s%s%s\n", the_event.event_name, the_event.event_date,
        the_event.event_time);
}

/*
 * Read event file data and add to struct
 */
void read_event_file(char filename[FILE_LENGTH]) {
    FILE *file;

    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Can't open file \"%s\"! (Does it exist?)\n",
         filename);
    } else {
        fgets(the_event.event_name,
           sizeof(the_event.event_name), file);
        fgets(the_event.event_date,
           sizeof(the_event.event_date), file);
        fgets(the_event.event_time,
           sizeof(the_event.event_time), file);
    }
    fclose(file);
}

/*
 * Updates the current time with paramenter
 */
void current_time(char time[TIME_LENGTH]) {
    strcpy(the_event.event_time, time);
}
```

1.2.3 nodes.c

```
/*
 * File:   nodes.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "runners.h"
#include "event_data.h"

static node *head = NULL;
static node *curr = NULL;

node the_node;

static node* create_list(int node_number,
 char node_type[NODE_TYPE_LENGTH]) {
    node *ptr = (node*)malloc(sizeof(node));
    if (NULL == ptr) {
        return NULL;
    }

    strcpy(ptr->node_type, node_type);
    ptr->node_number = node_number;
    ptr->next_node   = NULL;

    curr = ptr;
    head = ptr;
    return ptr;

}

static node* add_to_list(int node_number,
 char node_type[NODE_TYPE_LENGTH]) {
    if (NULL == head) {
        return (create_list(node_number, node_type));
    }

    node *ptr = (node*)malloc(sizeof(node));
    if (NULL == ptr) {
        return NULL;
    }

    strcpy(ptr->node_type, node_type);
    ptr->node_number = node_number;
    ptr->next_node   = NULL;

    curr->next_node  = ptr;
    curr = ptr;
    return ptr;
}

/*
 * Debug function to print checkpoints
 */
void print_node_list(void) {
    node *ptr = head;
    while (ptr != NULL) {
        printf("Node: %d Type: %s\n", ptr->node_number, ptr->node_type);
        ptr = ptr->next_node;
    }
    return;
}

/*
 * Read node file data and adds to linked list
 */
void read_node_file(char filename[FILE_LENGTH]) {
    FILE *file;
    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Can’t open the file \"%s\"!
         (Does it exist?)\n", filename);
    } else {
        node tmp_node;
        while (fscanf(file, "%d %s\n", &tmp_node.node_number,
            tmp_node.node_type) != EOF) {
            add_to_list(tmp_node.node_number, tmp_node.node_type);
        }
    }
}
```

1.2.4 tracks.c

```
/*
 * File:   tracks.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "runners.h"
#include "event_data.h"

static track *head = NULL;
static track *curr = NULL;

static track* create_list(int track_number, int track_start_node,
  int track_end_node, int track_avg_time) {
    track *ptr = (track*)malloc(sizeof(track));
    if (NULL == ptr) {
        return NULL;
    }

    ptr->track_number       = track_number;
    ptr->track_start_node   = track_start_node;
    ptr->track_end_node     = track_end_node;
    ptr->track_avg_time     = track_avg_time;
    ptr->next_track   = NULL;

    curr = ptr;
    head = ptr;
    return ptr;
}

static track* add_to_list(int track_number, int track_start_node,
  int track_end_node, int track_avg_time) {
    if (NULL == head) {
        return (create_list(track_number, track_start_node,
         track_end_node, track_avg_time));
    }

    track *ptr = (track*)malloc(sizeof(track));
    if (NULL == ptr) {
        return NULL;
    }

    ptr->track_number       = track_number;
    ptr->track_start_node   = track_start_node;
    ptr->track_end_node     = track_end_node;
    ptr->track_avg_time     = track_avg_time;
    ptr->next_track   = NULL;

    curr->next_track  = ptr;
    curr = ptr;
    return ptr;
}

/*
 * Debug function to print tracks
 */
void print_track_list(void) {
    track *ptr = head;
    while (ptr != NULL) {
        printf("Track: %d Start: %d End: %d Avg Time: %d\n",
         ptr->track_number, ptr->track_start_node,
         ptr->track_end_node, ptr->track_avg_time);
        ptr = ptr->next_track;
    }
    return;
}

/*
 * Read course track data and adds to linked list
 */
void read_track_file(char filename[FILE_LENGTH]) {
    FILE *file;
    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Can't open the file \"%s\"!
         (Does it exist?)\n", filename);
    } else {
        track tmp_track;
        while (fscanf(file, "%d %d %d %d\n", &tmp_track.track_number,
        &tmp_track.track_start_node, &tmp_track.track_end_node,
        &tmp_track.track_avg_time) != EOF) {
            add_to_list(tmp_track.track_number,
            tmp_track.track_start_node, tmp_track.track_end_node,
            tmp_track.track_avg_time);
        }
    }
}
```

1.2.5 courses.c

```
/*
 * File:   courses.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

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

```
#include "runners.h"
#include "event_data.h"

static course *head = NULL;
static course *curr = NULL;

static course* create_list(char course_identifier[COURSE_LENGTH],
   int course_number_of_nodes,
   int course_nodes[COURSE_NODE_MAX]) {
    course *ptr = (course*)malloc(sizeof(course));
    if (NULL == ptr) {
        return NULL;
    }

    strcpy(ptr->course_identifier, course_identifier);
    memcpy(ptr->course_nodes, course_nodes, sizeof(ptr->course_nodes));
    ptr->course_number_of_nodes = course_number_of_nodes;
    ptr->next_course            = NULL;

    curr = ptr;
    head = ptr;
    return ptr;
}

static course* add_to_list(char course_identifier[COURSE_LENGTH],
   int course_number_of_nodes,
   int course_nodes[COURSE_NODE_MAX]) {
    if (NULL == head) {
        return (create_list(course_identifier, course_number_of_nodes,
         course_nodes));
    }

    course *ptr = (course*)malloc(sizeof(course));
    if (NULL == ptr) {
        return NULL;
    }

    strcpy(ptr->course_identifier, course_identifier);
    memcpy(ptr->course_nodes, course_nodes, sizeof(ptr->course_nodes));
    ptr->course_number_of_nodes = course_number_of_nodes;
    ptr->next_course            = NULL;
    curr->next_course    = ptr;
    curr = ptr;
    return ptr;
}

/*
 * Debug function to print courses
 */
void print_course_list(void) {
    course *ptr = head;
    while (ptr != NULL) {
        printf("Course: %s Number of nodes: %d Nodes: ",
         ptr->course_identifier, ptr->course_number_of_nodes);
        int i;
        for (i=0;i<ptr->course_number_of_nodes;i++) {
            printf("%d ", ptr->course_nodes[i]);
        }
        printf("\n");
        ptr = ptr->next_course;
    }
    return;
}

/*
 * Read course file data and adds to linked list
 */
void read_course_file(char filename[FILE_LENGTH]) {
    FILE *file;
    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Can't open the file \"%s\"!
         (Does it exist?)\n", filename);
    } else {
        course tmp_course;
        char tmpstring[256];
        while (fscanf(file, "%s %d %[0-9 ]\n",
         tmp_course.course_identifier,
         &tmp_course.course_number_of_nodes, tmpstring) != EOF) {
            char * pch;
            pch = strtok(tmpstring, " ");
            int i;
            for (i=0;i<tmp_course.course_number_of_nodes;i++) {
                tmp_course.course_nodes[i] = atoi(pch);
                pch = strtok(NULL, " ");
            }
            add_to_list(tmp_course.course_identifier,
             tmp_course.course_number_of_nodes,
             tmp_course.course_nodes);
        }
    }
}

/*
 * Returns the starting checkpoint of a course
 */
int get_first_node(char course_identifier[COURSE_LENGTH]) {
    course *ptr = head;
    int first_node = 0;
    while (ptr != NULL) {
        if (strcmp(course_identifier, ptr->course_identifier) == 0) {
            first_node = ptr->course_nodes[0];
        }
        ptr = ptr->next_course;
    }
    return first_node;
}
```

1.2.6 entrants.c

```
/*
 * File:   entrants.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "runners.h"
#include "entrant_data.h"
#include "event_data.h"
#include "time_data.h"
```

```
static entrant *head = NULL;
static entrant *curr = NULL;

entrant the_entrant;

static entrant* create_list(int entrant_number,
char entrant_course[COURSE_LENGTH],
char entrant_name[NAME_LENGTH]) {
    entrant *ptr = (entrant*)malloc(sizeof(entrant));
    if (NULL == ptr) {
        return NULL;
    }

    strcpy(ptr->entrant_name, entrant_name);
    strcpy(ptr->entrant_course, entrant_course);
    strcpy(ptr->start_time, "");
    strcpy(ptr->end_time, "");
    ptr->entrant_number = entrant_number;
    ptr->last_seen      = 0;
    ptr->first_node     = get_first_node(ptr->entrant_course);
    ptr->next_entrant   = NULL;

    curr = ptr;
    head = ptr;
    return ptr;
}

static entrant* add_to_list(int entrant_number,
char entrant_course[COURSE_LENGTH], char entrant_name[NAME_LENGTH]) {
    if (NULL == head) {
        return (create_list(entrant_number, entrant_course,
         entrant_name));
    }

    entrant *ptr = (entrant*)malloc(sizeof(entrant));
    if (NULL == ptr) {
        return NULL;
    }

    strcpy(ptr->entrant_name, entrant_name);
    strcpy(ptr->entrant_course, entrant_course);
    strcpy(ptr->start_time, "");
    strcpy(ptr->end_time, "");
    ptr->entrant_number = entrant_number;
    ptr->last_seen      = 0;
    ptr->first_node     = get_first_node(ptr->entrant_course);
    ptr->next_entrant   = NULL;

    curr->next_entrant  = ptr;
    curr = ptr;
    return ptr;
}

/*
 * Debug function to print entrants
 */
void print_entrant_list(void) {
    entrant *ptr = head;
    while (ptr != NULL) {
        printf("Entrant Number: %d Course: %s Name: %s Last Seen: %d
         Start Time: %s End Time: %s\n", ptr->entrant_number,
         ptr->entrant_course, ptr->entrant_name, ptr->last_seen,
         ptr->start_time, ptr->end_time);
        ptr = ptr->next_entrant;
    }
}

/*
 * Read course entrant data and adds to linked list
 */
void read_entrant_file(char filename[FILE_LENGTH]) {
    FILE *file;
    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Can't open the file \"%s\"!
          (Does it exist?)\n", filename);
    } else {
        entrant tmp_entrant;
        while (fscanf(file, "%d %1s %[a-zA-Z ]\n",
        &tmp_entrant.entrant_number, tmp_entrant.entrant_course,
        tmp_entrant.entrant_name) != EOF) {
            add_to_list(tmp_entrant.entrant_number,
            tmp_entrant.entrant_course, tmp_entrant.entrant_name);
        }
    }
}

/*
 * Prints where a specified entrant was last seen and at what time
 */
void print_current_status(int entrant_number) {
    entrant *ptr = head;
    while (ptr != NULL) {
        if (ptr->entrant_number == entrant_number) {
            if (ptr->last_seen == 0) {
                printf("Entrant \"%s\" has not been seen at any
                checkpoint yet!\n", ptr->entrant_name);
            } else {
                printf("Entrant \"%s\" was last seen at node %d at
                %s.\n", ptr->entrant_name, ptr->last_seen, ptr->end_time);
            }
            break;
        } else {
            ptr = ptr->next_entrant;
        }
    }
}

/*
 * Updates the specified entrant's last seen location
 */
void update_entrant_location(int entrant_number, int location,
 char time[TIME_LENGTH]) {
    entrant *ptr = head;
    while (ptr != NULL) {
        if (ptr->entrant_number == entrant_number) {
            ptr->last_seen = location;
            if (strcmp(ptr->start_time, "") == 0) {
                strcpy(ptr->start_time, time);
            }
            strcpy(ptr->end_time, time);
            break;
        } else {
            ptr = ptr->next_entrant;
        }
    }
}

/*
 * Prints a list of entrants who have not yet started
 */
void print_unstarted_entrants(void) {
    entrant *ptr = head;
    printf("Entrants not started\n-----------------------------------------------
    printf("Competitor ID\tName\n------------------------------------------------
    while (ptr != NULL) {
        if (ptr->last_seen == 0) {
            printf("%d\t\t%s\n", ptr->entrant_number, ptr->entrant_name);
        }
        ptr = ptr->next_entrant;
    }
}

/*
 * Prints a list of entrants who are currently on the course
 */
void print_on_course_entrants(void) {
    entrant *ptr = head;
    printf("Entrants on course\n-------------------------------------------------
    printf("Competitor ID\tName\t\t\tCheckpoint\tTime\n---------------------------
    while (ptr != NULL) {
        if (ptr->last_seen != 0 && ptr->last_seen != ptr->first_node) {
            printf("%d\t\t%s\t\t%d\t%s\n", ptr->entrant_number,
             ptr->entrant_name, ptr->last_seen, ptr->end_time);
        }
        ptr = ptr->next_entrant;
    }
}

/*
 * Prints a list of finished entrants ordered by entrant ID
 */
void print_finished_entrants(void) {
    entrant *ptr = head;
    printf("Finished entrants\n--------------------------------------------------------
    printf("Competitor ID\tName\t\t\tFinished at\n----------------------------------
    while (ptr != NULL) {
        if (ptr->first_node == ptr->last_seen &&
         time_diff(ptr->start_time, ptr->end_time) > 0) {
            printf("%d\t\t%s\t\t%s\n", ptr->entrant_number,
             ptr->entrant_name, ptr->end_time);
        }
        ptr = ptr->next_entrant;
    }
}

/*
 * Prints a list of finished entrants ordered by time taken
 */
void print_results_table(void) {
    entrant *ptr = head;
    int totaltime = 0;
    printf("Results\n------------------------------------------------------------------
    printf("Competitor ID\tName\t\t\tTime Taken\n-----------------------------------
    while (ptr != NULL) {
        if (ptr->first_node == ptr->last_seen &&
         time_diff(ptr->start_time, ptr->end_time) > 0) {
            totaltime = time_diff(ptr->start_time, ptr->end_time);
            printf("%d\t\t%s\t\t%d Minutes\n", ptr->entrant_number,
             ptr->entrant_name, totaltime);
        }
        ptr = ptr->next_entrant;
    }
}
```

1.2.7 times.c

```
/*
 * File:   times.c
 * Author: Ben Brooks (beb12@aber.ac.uk)
 */

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

```
#include <string.h>
#include "runners.h"
#include "entrant_data.h"
#include "event_data.h"
#include "time_data.h"

/*
 * Reads time file data
 */
void read_time_file(char filename[FILE_LENGTH]) {
    FILE *file;
    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "Can’t open the file \"%s\"!
         (Does it exist?)\n", filename);
    } else {
        int node = 0;
        int competitor_id = 0;
        char type[TIME_TYPE_LENGTH];
        char time[TIME_LENGTH];
        while (fscanf(file, "%s %d %d %s\n", type, &node,
        &competitor_id, time) != EOF) {
            if (strcmp(type, "T") == 0) {
                update_entrant_location(competitor_id, node, time);
                current_time(time);
            } else {
                printf("An error occurred somewhere.
                 Is the time file syntax correct?\n");
            }
        }
    }
}

/*
 * Function to calculate difference between two times, in minutes
 */
int time_diff(char time1[TIME_LENGTH], char time2[TIME_LENGTH]) {
    int t1h, t1m, t2h, t2m;
    sscanf(time1, "%2d:%2d", &t1h, &t1m);
    sscanf(time2, "%2d:%2d", &t2h, &t2m);
    t1m += t1h*60;
    t2m += t2h*60;

    return t2m - t1m;
}
```

2 Compilation and Runtime output

2.1 Compilation Output

2.1.1 GNU C and C++ Compiler (gcc v3.4.3)

```
[beb12@minted:main]$ gcc -Wall *.c
main.c: In function ‘menu’:
main.c:74: warning: unused variable ‘diff’
```

2.1.2 Solaris Studio C Compiler (cc v5.12)

```
[beb12@minted:main]$ cc *.c
courses.c:
entrants.c:
event.c:
main.c:
nodes.c:
times.c:
tracks.c:
```

2.2 Runtime Output

```
[beb12@minted:main]$ ./a.out
Event file: ../../data/main/name.txt
Node file: ../../data/main/nodes.txt
Track file: ../../data/main/tracks.txt
Course file: ../../data/main/courses.txt
Entrant file: ../../data/main/entrants.txt
--------------------
Endurance Horse Race - Beginners Event
26th June 2012
07:30
--------------------
  1. Query the current location of a competitor
  2. List competitors which haven't started yet
  3. List competitors which are out on the course
  4. List competitors who have finished
  5. Manually supply times for a competitor
  6. Read in a file containing times
  7. Print a results list
  0. Quit
Choose an option: 6
Enter time file: ../../data/main/cp_times_1.txt
--------------------
Endurance Horse Race - Beginners Event
26th June 2012
08:54
--------------------
  1. Query the current location of a competitor
  ~~ truncated ~~
  0. Quit
Choose an option: 7
Results
-----------------------------------------------------------------
Competitor ID   Name                    Time Taken
-----------------------------------------------------------------
--------------------
Endurance Horse Race - Beginners Event
26th June 2012
08:54
--------------------
  1. Query the current location of a competitor
  ~~ truncated ~~
  0. Quit
Choose an option: 6
Enter time file: ../../data/main/cp_times_2.txt
--------------------
Endurance Horse Race - Beginners Event
26th June 2012
10:33
--------------------
  1. Query the current location of a competitor
  ~~ truncated ~~
  0. Quit
Choose an option: 7
```

```
Results
-----------------------------------------------------------------
Competitor ID   Name                    Time Taken
-----------------------------------------------------------------
1               Donald Duck             121 Minutes
2               Mickey Mouse            119 Minutes
3               Jemima Julieta Mouse            123 Minutes
4               Minnie Duck             123 Minutes
5               Minnie Mouse            138 Minutes
6               Minnie Mouse Junior             127 Minutes
7               Deputy Doug             126 Minutes
8               Deputy Duck             130 Minutes
9               Bewick Swan             111 Minutes
10              Black Swan              119 Minutes
11              Albert Einstein         132 Minutes
12              Albert Mouse            125 Minutes
13              Donald Duck Senior              131 Minutes
14              Egbert Einstein         114 Minutes
--------------------
Endurance Horse Race - Beginners Event
26th June 2012
10:33
--------------------
  1. Query the current location of a competitor
  ~~ truncated ~~
  0. Quit
Choose an option: 0
Bye Bye!
```

3 Testing

ID	Description	Input	Output	Expected	Pass
1	Enter a non-existent file	qwerty	Can't open file "qwerty"! (Does it exist?)	Can't open file "asd"! (Does it exist?)	Yes
2	Enter an existing file	nodes.txt	No output	No output	Yes
3	Enter an invalid menu choice	h	Bye Bye!	Bye Bye!	Yes
3	Enter an invalid menu choice	8	Invalid Selection!	Invalid Selection!	Yes
4	Enter a valid menu choice	1	Choose an option: 1 Enter Entrant ID: 3	Choose an option: 1 Enter Entrant ID: 3	Yes
4	Quit program	0	Bye Bye!	Bye Bye!	Yes

4 Design Decisions

Before starting this assignment I had a rough idea of how I would read in the data files and store them. I had planned on using linked lists for everything as it allows for greater flexibility with regards to data set sizes than arrays when using C. I started off by reading in the first data file, name.txt. This contains the name, date and starting time of the event. From there I could then use structures for nodes, tracks, courses and entrants.

Most of my C files contain the same basic structure for a linked list, they have a few static functions for adding to/creating a list and a few external functions for manipulating the data stored in the list. I feel like this could have probably been achieved in a more efficient way with less code duplication, but given time constraints and my unfamiliarity with C I felt more comfortable doing it the way I did.

5 Conclusion

I feel I had a very reasonable go at the main mission, and attempted some of the extended mission with what little time I had left. I ensured my program compiles without any warnings in both the GNU and Solaris compilers and ran without any segfaults or strange errors.

Throughout development, I used a private Git repository hosted on GitHub. This ensured the entirety of my program was version controlled and I was able to revert back to a previous revision if things went wrong. Just after I finalise my handin, I'm going to create a branch so that I can continue to work on this program after it has been submitted for my own personal project. After the assignment is marked, I plan on making this repository publicly viewable and serve as a portfolio item.

References

[1] P. Prinz and U. Kirch-Prinz *The C Pocket Reference* 2002

[2] Prinz and Crawford *C In a Nutshell* 2005

[3] Singly linked lists in C *http://www.cprogramming.com/tutorial/c/lesson15.html*