

## CS23710 Assignment

Generated by Doxygen 1.8.1.2

Fri Dec 14 2012 10:33:17



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	course Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Field Documentation . . . . .	5
3.1.2.1	course_id . . . . .	5
3.2	course_list Struct Reference . . . . .	6
3.2.1	Detailed Description . . . . .	6
3.2.2	Field Documentation . . . . .	6
3.2.2.1	entrants . . . . .	6
3.2.2.2	node_list . . . . .	6
3.2.2.3	track_list . . . . .	6
3.3	cp_time Struct Reference . . . . .	7
3.3.1	Detailed Description . . . . .	7
3.4	cp_time_list Struct Reference . . . . .	7
3.4.1	Detailed Description . . . . .	7
3.5	entrant Struct Reference . . . . .	7
3.5.1	Detailed Description . . . . .	8
3.5.2	Field Documentation . . . . .	8
3.5.2.1	status . . . . .	8
3.6	entrant_list Struct Reference . . . . .	8
3.6.1	Detailed Description . . . . .	9
3.7	event Struct Reference . . . . .	9
3.7.1	Detailed Description . . . . .	9
3.7.2	Field Documentation . . . . .	10
3.7.2.1	date . . . . .	10
3.7.2.2	entrants . . . . .	10

3.7.2.3	name	10
3.8	node Struct Reference	10
3.8.1	Detailed Description	10
3.8.2	Field Documentation	11
3.8.2.1	node_id	11
3.9	node_list Struct Reference	11
3.9.1	Detailed Description	11
3.10	time_struct Struct Reference	11
3.10.1	Detailed Description	12
3.11	track Struct Reference	12
3.11.1	Detailed Description	12
3.12	track_list Struct Reference	12
3.12.1	Detailed Description	13
<b>4</b>	<b>File Documentation</b>	<b>15</b>
4.1	course.c File Reference	15
4.1.1	Detailed Description	15
4.1.2	Function Documentation	16
4.1.2.1	read_courses	16
4.1.2.2	search_course_id	16
4.1.2.3	validate_course	16
4.2	course.h File Reference	16
4.2.1	Detailed Description	17
4.2.2	Typedef Documentation	17
4.2.2.1	COURSE	17
4.2.2.2	COURSE_LIST	17
4.2.3	Function Documentation	18
4.2.3.1	read_courses	18
4.2.3.2	search_course_id	18
4.2.3.3	validate_course	18
4.3	cp_time.c File Reference	19
4.3.1	Detailed Description	19
4.3.2	Function Documentation	19
4.3.2.1	read_event_data	19
4.4	cp_time.h File Reference	19
4.4.1	Detailed Description	20
4.5	entrant.c File Reference	20
4.5.1	Detailed Description	21
4.5.2	Function Documentation	21
4.5.2.1	entrant_id_search	21

4.5.2.2	<a href="#">entrant_name_search</a>	21
4.5.2.3	<a href="#">read_entrants</a>	21
4.5.2.4	<a href="#">update_status</a>	22
4.6	<a href="#">entrant.h File Reference</a>	22
4.6.1	<a href="#">Detailed Description</a>	23
4.6.2	<a href="#">Typedef Documentation</a>	23
4.6.2.1	<a href="#">ENTRANT_LIST</a>	23
4.6.3	<a href="#">Function Documentation</a>	23
4.6.3.1	<a href="#">entrant_id_search</a>	23
4.6.3.2	<a href="#">entrant_name_search</a>	23
4.6.3.3	<a href="#">print_entrant</a>	24
4.6.3.4	<a href="#">read_entrants</a>	24
4.7	<a href="#">event.c File Reference</a>	24
4.7.1	<a href="#">Detailed Description</a>	25
4.7.2	<a href="#">Function Documentation</a>	25
4.7.2.1	<a href="#">read_event</a>	25
4.8	<a href="#">event.h File Reference</a>	25
4.8.1	<a href="#">Detailed Description</a>	26
4.8.2	<a href="#">Typedef Documentation</a>	26
4.8.2.1	<a href="#">EVENT</a>	26
4.8.3	<a href="#">Function Documentation</a>	26
4.8.3.1	<a href="#">read_event</a>	26
4.9	<a href="#">functions.c File Reference</a>	26
4.9.1	<a href="#">Detailed Description</a>	27
4.9.2	<a href="#">Function Documentation</a>	27
4.9.2.1	<a href="#">add_cp</a>	27
4.9.2.2	<a href="#">calc_time</a>	27
4.9.2.3	<a href="#">entrant_finished</a>	28
4.9.2.4	<a href="#">finished</a>	28
4.9.2.5	<a href="#">format_time</a>	28
4.9.2.6	<a href="#">not_started</a>	28
4.9.2.7	<a href="#">out_track</a>	29
4.9.2.8	<a href="#">print_all</a>	29
4.9.2.9	<a href="#">print_entrant</a>	29
4.10	<a href="#">functions.h File Reference</a>	29
4.10.1	<a href="#">Detailed Description</a>	30
4.10.2	<a href="#">Function Documentation</a>	30
4.10.2.1	<a href="#">add_cp</a>	30
4.10.2.2	<a href="#">calc_time</a>	30
4.10.2.3	<a href="#">entrant_finished</a>	30

4.10.2.4	finished	31
4.10.2.5	format_time	31
4.10.2.6	not_started	31
4.10.2.7	out_track	31
4.10.2.8	print_all	31
4.10.2.9	read_event_data	32
4.10.2.10	read_tracks	32
4.10.2.11	update_status	32
4.11	input.c File Reference	32
4.11.1	Detailed Description	33
4.11.2	Function Documentation	33
4.11.2.1	ask_int	33
4.11.2.2	ask_str	34
4.11.2.3	get_courses	34
4.11.2.4	get_cp_data	34
4.11.2.5	get_entrants	34
4.11.2.6	get_event	35
4.11.2.7	get_nodes	35
4.11.2.8	get_tracks	35
4.11.2.9	load_cp_data	35
4.11.2.10	query_entrant	35
4.11.2.11	show_menu	36
4.12	input.h File Reference	36
4.12.1	Detailed Description	37
4.12.2	Function Documentation	37
4.12.2.1	ask_int	37
4.12.2.2	ask_str	37
4.12.2.3	get_courses	37
4.12.2.4	get_cp_data	38
4.12.2.5	get_entrants	38
4.12.2.6	get_event	38
4.12.2.7	get_nodes	38
4.12.2.8	get_tracks	38
4.12.2.9	load_cp_data	39
4.12.2.10	query_entrant	39
4.12.2.11	show_menu	39
4.13	node.c File Reference	39
4.13.1	Detailed Description	40
4.13.2	Function Documentation	40
4.13.2.1	node_id_search	40

4.13.2.2	<a href="#">read_nodes</a>	40
4.14	<a href="#">node.h File Reference</a>	40
4.14.1	<a href="#">Detailed Description</a>	41
4.14.2	<a href="#">Typedef Documentation</a>	41
4.14.2.1	<a href="#">NODE</a>	41
4.14.2.2	<a href="#">NODE_LIST</a>	41
4.14.3	<a href="#">Function Documentation</a>	42
4.14.3.1	<a href="#">node_id_search</a>	42
4.14.3.2	<a href="#">read_nodes</a>	42
4.15	<a href="#">tests.h File Reference</a>	42
4.15.1	<a href="#">Detailed Description</a>	43
4.15.2	<a href="#">Function Documentation</a>	43
4.15.2.1	<a href="#">course_read</a>	43
4.16	<a href="#">time_struct.h File Reference</a>	43
4.16.1	<a href="#">Detailed Description</a>	43
4.17	<a href="#">track.c File Reference</a>	44
4.17.1	<a href="#">Detailed Description</a>	44
4.17.2	<a href="#">Function Documentation</a>	44
4.17.2.1	<a href="#">read_tracks</a>	44
4.17.2.2	<a href="#">validate_track</a>	45
4.18	<a href="#">track.h File Reference</a>	45
4.18.1	<a href="#">Detailed Description</a>	45
4.18.2	<a href="#">Typedef Documentation</a>	46
4.18.2.1	<a href="#">TRACK</a>	46
4.18.2.2	<a href="#">TRACK_LIST</a>	46
4.18.3	<a href="#">Function Documentation</a>	46
4.18.3.1	<a href="#">validate_track</a>	46





# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">course</a>	This data structure holds information about a course in the event . . . . .	5
<a href="#">course_list</a>	This is the course list . . . . .	6
<a href="#">cp_time</a>	This data structure represents a check-in at a checkpoint . . . . .	7
<a href="#">cp_time_list</a>	This structure is used to keep the list of CP_TIME elements . . . . .	7
<a href="#">entrant</a>	A typedef of a struct representing a entrant in the competition . . . . .	7
<a href="#">entrant_list</a>	This is the entrant list . . . . .	8
<a href="#">event</a>	The event structure holds information about the event such as the name, date, and time . . . .	9
<a href="#">node</a>	The node data structure represents a checkpoint in the course . . . . .	10
<a href="#">node_list</a>	This is the node list . . . . .	11
<a href="#">time_struct</a>	The structure used to represent time . . . . .	11
<a href="#">track</a>	This structure holds the information about each track . . . . .	12
<a href="#">track_list</a>	This is the track list . . . . .	12



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">course.c</a>	This file contains functions used to work with the course data structure . . . . .	15
<a href="#">course.h</a>	This file contains the definition of the course data structure and definitions of functions used to work with the data structure . . . . .	16
<a href="#">cp_time.c</a>	This file contains functions used to work with the CP_TIME data structure . . . . .	19
<a href="#">cp_time.h</a>	This file contains the definition of the data structure used to represent a entrant checking in at a check point . . . . .	19
<a href="#">entrant.c</a>	This file holds the functions used to work with the ENTRANT data structure . . . . .	20
<a href="#">entrant.h</a>	This file holds structures and values needed to work with entrants . . . . .	22
<a href="#">event.c</a>	This file contains function used to work with the event data structure . . . . .	24
<a href="#">event.h</a>	This file contains the definition of the event structure and other variables related to the event . . . . .	25
<a href="#">functions.c</a>	This file contains various functions for the program . . . . .	26
<a href="#">functions.h</a>	This file contains definitions of various functions . . . . .	29
<a href="#">input.c</a>	This file holds functions related to getting input from the user . . . . .	32
<a href="#">input.h</a>	This file contains function definitions for <a href="#">input.c</a> . . . . .	36
<a href="#">node.c</a>	This file holds function used to wok with the node data structure . . . . .	39
<a href="#">node.h</a>	This file contains the definition of the node data structure and definitions of functions used to work with the data structure . . . . .	40
<a href="#">tests.h</a>	This file contains the definitions of several functi . . . . .	42
<a href="#">time_struct.h</a>	This file contains the definition of the data structure used to represent the time structure . . . . .	43
<a href="#">track.c</a>	This file contains functions used to work with the track data structure . . . . .	44

[track.h](#)

This file contains the definition of the track structure . . . . . [45](#)

## Chapter 3

# Data Structure Documentation

### 3.1 course Struct Reference

This data structure holds information about a course in the event.

```
#include <course.h>
```

#### Data Fields

- char [course\\_id](#)  
*The course id.*
- int [nodes\\_size](#)  
*The number of nodes in the course.*
- struct [course](#) \* [next](#)  
*Pointer to the next element in the list.*
- int [time\\_cp](#)  
*Number of check points where time is recorded in the course.*
- int [elements](#)  
*Number of node elements currently in the nodes array.*
- int [nodes](#) [100]  
*Array of nodes in the course.*

#### 3.1.1 Detailed Description

This data structure holds information about a course in the event.

A course consists of several nodes and a single capital letter to identify the course. This data structure is also used as a linked list, and so has a pointer to the next element in the list.

#### 3.1.2 Field Documentation

##### 3.1.2.1 char course::course\_id

The course id.

The id is a single capital letter character.

The documentation for this struct was generated from the following file:

- [course.h](#)

## 3.2 course\_list Struct Reference

This is the course list.

```
#include <course.h>
```

### Data Fields

- [COURSE \\* head](#)  
*The head of the list.*
- [COURSE \\* tail](#)  
*The tail of the list.*
- [NODE\\_LIST \\* node\\_list](#)  
*A pointer to a NODE\_LIST containing nodes that are used in this course.*
- [TRACK\\_LIST \\* track\\_list](#)  
*A pointer to a TRACK\_LIST containing tracks that are used in this course.*
- [ENTRANT \\* entrants](#)  
*A pointer to a ENTRANT array containing entrants that are participating in this course.*
- [int size](#)  
*The size of the course list.*

### 3.2.1 Detailed Description

This is the course list.

It is used to easily pass around the list and information about it to make working with it easier.

### 3.2.2 Field Documentation

#### 3.2.2.1 ENTRANT\* course\_list::entrants

A pointer to a ENTRANT array containing entrants that are participating in this course.

The list may also contain entrants that are not participating in this course.

#### 3.2.2.2 NODE\_LIST\* course\_list::node\_list

A pointer to a NODE\_LIST containing nodes that are used in this course.

The list may also contain nodes that are not used in this course.

#### 3.2.2.3 TRACK\_LIST\* course\_list::track\_list

A pointer to a TRACK\_LIST containing tracks that are used in this course.

The list may also contain tracks that are not used in this course.

The documentation for this struct was generated from the following file:

- [course.h](#)

## 3.3 cp\_time Struct Reference

This data structure represents a check-in at a checkpoint.

```
#include <cp_time.h>
```

### Data Fields

- `NODE * node`  
*Pointer to the node that represents the checkpoint.*
- `char status`  
*The status of the entrant when he reaches the check point.*
- `TIME_STRUCT * time`  
*The string containing the time the entrant reached the CP.*
- `struct cp_time * next`  
*Pointer to the next element in the linked list.*

#### 3.3.1 Detailed Description

This data structure represents a check-in at a checkpoint.

The documentation for this struct was generated from the following file:

- `cp_time.h`

## 3.4 cp\_time\_list Struct Reference

This structure is used to keep the list of CP\_TIME elements.

```
#include <cp_time.h>
```

### Data Fields

- `CP_TIME * head`  
*Pointer to the head of the list.*
- `CP_TIME * tail`  
*Pointer to the tail of the list.*
- `int size`  
*Number of elements in the list.*

#### 3.4.1 Detailed Description

This structure is used to keep the list of CP\_TIME elements.

The documentation for this struct was generated from the following file:

- `cp_time.h`

## 3.5 entrant Struct Reference

A typedef of a struct representing a entrant in the competition.

```
#include <entrant.h>
```

## Data Fields

- int [comp\\_number](#)  
*The competitor number for the entrant.*
- char [course\\_id](#)  
*The Single letter course code for the course the entrant is participating in.*
- int [excluded](#)  
*Boolean used to denote if the entrant has been excluded from the race.*
- char [name](#) [MAX\_NAME\_LENGTH]  
*The name of the entrant.*
- [NODE](#) \* [current\\_node](#)  
*The node the entrant currently is at.*
- [TIME\\_STRUCT](#) \* [start\\_time](#)  
*The time the entrant reached the first check point.*
- char [total\\_time](#) [20]  
*The time the entrant has used so far.*
- char [status](#) [20]  
*The current status of the entrant.*
- [CP\\_TIME](#) \* [last\\_cp](#)  
*Last check point the entrant was at.*
- struct [entrant](#) \* [next](#)  
*A pointer to the next element in the linked list.*
- [CP\\_LIST](#) \* [checkpoints](#)  
*List of check points the entrant has been at so far.*

### 3.5.1 Detailed Description

A typedef of a struct representing a entrant in the competition.

### 3.5.2 Field Documentation

#### 3.5.2.1 char entrant::status[20]

The current status of the entrant.

It can either be "Not Started", "Finished" or the id of the node the entrant last checked in at.

The documentation for this struct was generated from the following file:

- [entrant.h](#)

## 3.6 entrant\_list Struct Reference

This is the entrant list.

```
#include <entrant.h>
```



## Data Fields

- [ENTRANT](#) \* [head](#)  
*Pointer to the head of the linked list.*
- [ENTRANT](#) \* [tail](#)  
*Pointer to the tail of the linked list.*
- int [size](#)  
*The size of the linked list.*

### 3.6.1 Detailed Description

This is the entrant list.

It is used to easily pass around the list and information about it to make working with it easier.

The documentation for this struct was generated from the following file:

- [entrant.h](#)

## 3.7 event Struct Reference

The event structure holds information about the event such as the name, date, and time.

```
#include <event.h>
```

## Data Fields

- char [name](#) [80]  
*The name of the event.*
- char [date](#) [80]  
*The date of the event.*
- char [time](#) [80]  
*The time of the event.*
- [ENTRANT\\_LIST](#) \* [entrants](#)  
*An array of the entrants participating in the event.*
- [NODE\\_LIST](#) \* [nodes](#)  
*Linked list of all the courses in the event.*
- [TRACK\\_LIST](#) \* [tracks](#)  
*List of all the tracks in this event.*
- [COURSE\\_LIST](#) \* [courses](#)  
*List of all the courses in this event.*

### 3.7.1 Detailed Description

The event structure holds information about the event such as the name, date, and time.

The structure also has a array of entrants and a linked list of the course nodes.

### 3.7.2 Field Documentation

#### 3.7.2.1 char event::date[80]

The date of the event.

The date is in free text format, and no longer than 80 characters.

#### 3.7.2.2 ENTRANT\_LIST\* event::entrants

An array of the entrants participating in the event.

The size of the array is dynamic.

#### 3.7.2.3 char event::name[80]

The name of the event.

Name can not be longer than 80 characters.

The documentation for this struct was generated from the following file:

- [event.h](#)

## 3.8 node Struct Reference

The node data structure represents a checkpoint in the course.

```
#include <node.h>
```

### Data Fields

- int [node\\_id](#)  
*The node id used to identify the node.*
- char [node\\_type](#) [10]  
*The node type.*
- struct [node](#) \* [next](#)  
*Pointer to the next node element.*
- int [elements](#)  
*Number of track elements currently in the tracks array.*
- [TRACK](#) [tracks](#) [100]  
*Array of tracks for this node.*

### 3.8.1 Detailed Description

The node data structure represents a checkpoint in the course.

There are several different checkpoints. Each node is identified by a positive integer. The data structure is also a linked list and has a pointer to the next element in the list.

### 3.8.2 Field Documentation

#### 3.8.2.1 int node::node\_id

The node id used to identify the node.

The id is a positive integer.

The documentation for this struct was generated from the following file:

- [node.h](#)

## 3.9 node\_list Struct Reference

This is the node list.

```
#include <node.h>
```

### Data Fields

- [NODE \\* head](#)  
*The head of the linked list.*
- [NODE \\* tail](#)  
*The tail of the linked list.*
- int [size](#)  
*The size of the linked list.*

### 3.9.1 Detailed Description

This is the node list.

It is used to easily pass around the list and information about it to make working with it easier.

The documentation for this struct was generated from the following file:

- [node.h](#)

## 3.10 time\_struct Struct Reference

The structure used to represent time.

```
#include <time_struct.h>
```

### Data Fields

- int **hours**
- int **minutts**
- char [time\\_str](#) [20]  
*The current time in string format: xH yM.*

### 3.10.1 Detailed Description

The structure used to represent time.

The documentation for this struct was generated from the following file:

- [time\\_struct.h](#)

## 3.11 track Struct Reference

This structure holds the information about each track.

```
#include <track.h>
```

### Data Fields

- int [track\\_id](#)  
*The ID of the track.*
- int [start\\_id](#)  
*The ID of the start node.*
- int [end\\_id](#)  
*The ID of the end node.*
- int [time](#)  
*The estimated time a entrant should use to go from start to end.*
- struct [track](#) \* [next](#)  
*Pointer to the next track element in the linked list.*

### 3.11.1 Detailed Description

This structure holds the information about each track.

The structure also has a pointer to the another track to act as a linked list.

The documentation for this struct was generated from the following file:

- [track.h](#)

## 3.12 track\_list Struct Reference

This is the track list.

```
#include <track.h>
```

### Data Fields

- [TRACK](#) \* [head](#)  
*Head of the list.*
- [TRACK](#) \* [tail](#)  
*Tail of the list.*
- int [size](#)  
*Size of the list.*

### 3.12.1 Detailed Description

This is the track list.

It is used to easily pass around the list and information about it to make working with it easier.

The documentation for this struct was generated from the following file:

- [track.h](#)



# Chapter 4

## File Documentation

### 4.1 course.c File Reference

This file contains functions used to work with the course data structure.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include "node.h"
#include "track.h"
#include "course.h"
#include "entrant.h"
```

#### Functions

- `COURSE_LIST * read_courses` (char \*path, `NODE_LIST *node_list`, `TRACK_LIST *track_list`)  
*This function reads and parses the file with the courses.*
- int `validate_course` (`COURSE course`, `NODE_LIST *node_list`)  
*Validates if the COURSE is combined of nodes with that tracks that are valid.*
- `COURSE * search_course_id` (`COURSE_LIST *list`, char id)  
*This function searches for a COURSE with the given ID.*

#### Variables

- int **errno**

#### 4.1.1 Detailed Description

This file contains functions used to work with the course data structure.

##### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

##### Date

2012.11.25

## 4.1.2 Function Documentation

### 4.1.2.1 `COURSE_LIST* read_courses ( char * path, NODE_LIST * node_list, TRACK_LIST * track_list )`

This function reads and parses the file with the courses.

The function will check that the node specified exists, and that there is a valid track between the nodes. The data should consist of a single capital character which will be the course ID. Followed by a positive integer which will be the number of nodes for the course. The rest of the line should be the ID of each node separated by spaces.

#### Parameters

<i>path</i>	Path to the file containing the course data.
<i>node_list</i>	Pointer to the node list. It is needed to verify that the nodes in the course exists.
<i>track_list</i>	Pointer to the track list. It is needed to verify that there is a valid path between the nodes of the course.

#### Returns

If the function is successful it will return a pointer to the first element of the list. If it fails it will return NULL.

### 4.1.2.2 `COURSE* search_course_id ( COURSE_LIST * list, char id )`

This function searches for a COURSE with the given ID.

#### Parameters

<i>list</i>	List of the courses.
<i>id</i>	The ID of the COURSE you are looking for.

#### Returns

Returns a pointer to the COURSE if it is found, if it fails NULL is returned.

### 4.1.2.3 `int validate_course ( COURSE course, NODE_LIST * node_list )`

Validates if the COURSE is combined of nodes with that tracks that are valid.

#### Parameters

<i>course</i>	The course you want to validate.
<i>node_list</i>	A list of nodes, this is needed to lookup the nodes.

#### Returns

Returns 1 if it is a valid COURSE. If the COURSE is invalid 0 is returned.

## 4.2 course.h File Reference

This file contains the definition of the course data structure and definitions of functions used to work with the data structure.

```
#include "entrant.h"
#include "node.h"
#include "track.h"
```



## Data Structures

- struct `course`  
*This data structure holds information about a course in the event.*
- struct `course_list`  
*This is the course list.*

## Typedefs

- typedef struct `course` `COURSE`  
*This data structure holds information about a course in the event.*
- typedef struct `course_list` `COURSE_LIST`  
*This is the course list.*

## Functions

- int `validate_course` (`COURSE` `course`, `NODE_LIST` \*`node_list`)  
*Validates if the COURSE is combined of nodes with that tracks that are valid.*
- `COURSE_LIST` \* `read_courses` (`char` \*`path`, `NODE_LIST` \*`node_list`, `TRACK_LIST` \*`track_list`)  
*This function reads and parses the file with the courses.*
- `COURSE` \* `search_course_id` (`COURSE_LIST` \*`list`, `char` `id`)  
*This function searches for a COURSE with the given ID.*

### 4.2.1 Detailed Description

This file contains the definition of the course data structure and definitions of functions used to work with the data structure.

#### Author

Sindre Smistad `sis13@aber.ac.uk`

#### Date

2012.11.25

### 4.2.2 Typedef Documentation

#### 4.2.2.1 typedef struct `course` `COURSE`

This data structure holds information about a course in the event.

A course consists of several nodes and a single capital letter to identify the course. This data structure is also used as a linked list, and so has a pointer to the next element in the list.

#### 4.2.2.2 typedef struct `course_list` `COURSE_LIST`

This is the course list.

It is used to easily pass around the list and information about it to make working with it easier.

### 4.2.3 Function Documentation

#### 4.2.3.1 **COURSE\_LIST\*** read\_courses ( *char \* path*, **NODE\_LIST \*** *node\_list*, **TRACK\_LIST \*** *track\_list* )

This function reads and parses the file with the courses.

The function will check that the node specified exists, and that there is a valid track between the nodes. The data should consist of a single capital character which will be the course ID. Followed by a positive integer which will be the number of nodes for the course. The rest of the line should be the ID of each node separated by spaces.

##### Parameters

<i>path</i>	Path to the file containing the course data.
<i>node_list</i>	Pointer to the node list. It is needed to verify that the nodes in the course exists.
<i>track_list</i>	Pointer to the track list. It is needed to verify that there is a valid path between the nodes of the course.

##### Returns

If the function is successful it will return a pointer to the first element of the list. If it fails it will return NULL.

#### 4.2.3.2 **COURSE\*** search\_course\_id ( **COURSE\_LIST \*** *list*, *char id* )

This function searches for a COURSE with the given ID.

##### Parameters

<i>list</i>	List of the courses.
<i>id</i>	The ID of the COURSE you are looking for.

##### Returns

Returns a pointer to the COURSE if it is found, if it fails NULL is returned.

#### 4.2.3.3 **int** validate\_course ( **COURSE** *course*, **NODE\_LIST \*** *node\_list* )

Validates if the COURSE is combined of nodes with that tracks that are valid.

##### Parameters

<i>course</i>	The course you want to validate.
<i>node_list</i>	A list of nodes, this is needed to lookup the nodes.

### Returns

Returns 1 if it is a valid COURSE. If the COURSE is invalid 0 is returned.

## 4.3 cp\_time.c File Reference

This file contains functions used to work with the CP\_TIME data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include "cp_time.h"
#include "node.h"
#include "event.h"
#include "functions.h"
```

### Functions

- int [read\\_event\\_data](#) (char \*path, [EVENT](#) \*event)  
*This function reads the check point data from a file.*

#### 4.3.1 Detailed Description

This file contains functions used to work with the CP\_TIME data structure.

### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

### Date

2012.12.06

#### 4.3.2 Function Documentation

##### 4.3.2.1 int read\_event\_data ( char \* path, EVENT \* event )

This function reads the check point data from a file.

### Parameters

<i>path</i>	Path to the file containing the checkpoint data.
<i>event</i>	Event structure.

### Returns

On success this function returns 1, 0 is returned on failure.

## 4.4 cp\_time.h File Reference

This file contains the definition of the data structure used to represent a entrant checking in at a check point.

```
#include "node.h"
#include "time_struct.h"
```

## Data Structures

- struct [cp\\_time](#)  
*This data structure represents a check-in at a checkpoint.*
- struct [cp\\_time\\_list](#)  
*This structure is used to keep the list of CP\_TIME elements.*

## Typedefs

- typedef struct [cp\\_time](#) CP\_TIME  
*This data structure represents a check-in at a checkpoint.*
- typedef struct [cp\\_time\\_list](#) CP\_LIST  
*This structure is used to keep the list of CP\_TIME elements.*

### 4.4.1 Detailed Description

This file contains the definition of the data structure used to represent a entrant checking in at a check point.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.12.06

## 4.5 entrant.c File Reference

This file holds the functions used to work with the ENTRANT data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include "entrant.h"
#include "event.h"
#include "course.h"
#include "node.h"
#include "functions.h"
```

## Functions

- [ENTRANT\\_LIST](#) \* [read\\_entrants](#) (char \*path)  
*This function opens a file containing the data about the entrants.*
- void [update\\_status](#) ([ENTRANT](#) \*ent, [EVENT](#) \*event)  
*Updates the status of entrant.*
- [ENTRANT](#) \* [entrant\\_id\\_search](#) (int entrant\_id, [ENTRANT\\_LIST](#) \*entrants)

*Searches the entrant list for a entrant that has the passed ID.*

- **ENTRANT** \* **entrant\_name\_search** (char \*name, **ENTRANT\_LIST** \*entrants)

*Search the entrant list by name.*

## Variables

- int **errno**

### 4.5.1 Detailed Description

This file holds the functions used to work with the ENTRANT data structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.20

### 4.5.2 Function Documentation

#### 4.5.2.1 **ENTRANT**\* **entrant\_id\_search** ( int *entrant\_id*, **ENTRANT\_LIST** \* *entrants* )

Searches the entrant list for a entrant that has the passed ID.

##### Parameters

<i>entrant_id</i>	The ID of the entrant to search for.
<i>entrants</i>	List of entrants.

##### Returns

Returns a pointer to the entrant if found. If no entrant is found NULL is returned.

#### 4.5.2.2 **ENTRANT**\* **entrant\_name\_search** ( char \* *name*, **ENTRANT\_LIST** \* *entrants* )

Search the entrant list by name.

##### Parameters

<i>name</i>	Name of the entrant you are searching for.
<i>entrants</i>	List of entrants.

##### Returns

Returns a pointer to the entrant if found. If no entrant is found NULL is returned.

#### 4.5.2.3 **ENTRANT\_LIST**\* **read\_entrants** ( char \* *path* )

This function opens a file containing the data about the entrants.

The function does some simple error checking on the file, and will return a negative integer on failure. The Data

should include a positive integer that represents a id. A single capital character which is the id of the course the entrant has signed up for. Followed by a no longer than 80 characters long name.

#### Parameters

<i>path</i>	Path to the file you want to open.
-------------	------------------------------------

#### Returns

Returns 1 if successful, a negative integer otherwise.

#### 4.5.2.4 void update\_status ( ENTRANT \* *ent*, EVENT \* *event* )

Updates the status of entrant.

Valid statuses are "Not Started", "Finished", or the id of the last node the entrant checked in at.

#### Parameters

<i>ent</i>	The entrant to update.
<i>event</i>	Event structure.

## 4.6 entrant.h File Reference

This file holds structures and values needed to work with entrants.

```
#include "cp_time.h"
#include "time_struct.h"
```

### Data Structures

- struct [entrant](#)  
*A typedef of a struct representing a entrant in the competition.*
- struct [entrant\\_list](#)  
*This is the entrant list.*

### Macros

- #define [MAX\\_NAME\\_LENGTH](#) 50  
*As stated in assignment, no name is longer than 50 chars including spaces.*
- #define [ENTRANTS\\_MIN\\_SIZE](#) 100  
*Minimum size of the entrant array.*
- #define [ENT\\_FORMAT](#) "%-5d%-5s%-10c%-15s%-15s%-10s\n"  
*The format string used when printing out data about a entrant.*

### Typedefs

- typedef struct [entrant](#) [ENTRANT](#)  
*A typedef of a struct representing a entrant in the competition.*
- typedef struct [entrant\\_list](#) [ENTRANT\\_LIST](#)  
*This is the entrant list.*

## Functions

- `ENTRANT_LIST * read_entrants (char *path)`  
*This function opens a file containing the data about the entrants.*
- `ENTRANT * entrant_id_search (int entrant_id, ENTRANT_LIST *entrants)`  
*Searches the entrant list for a entrant that has the passed ID.*
- `void print_entrant (ENTRANT *entrant)`  
*Prints out nicely formatted info about a entrant.*
- `void print_entrant_header ()`  
*Prints out the header used showing what each colum is.*
- `ENTRANT * entrant_name_search (char *name, ENTRANT_LIST *entrants)`  
*Search the entrant list by name.*

### 4.6.1 Detailed Description

This file holds structures and values needed to work with entrants.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.20

### 4.6.2 Typedef Documentation

#### 4.6.2.1 typedef struct entrant\_list ENTRANT\_LIST

This is the entrant list.

It is used to easily pass around the list and information about it to make working with it easier.

### 4.6.3 Function Documentation

#### 4.6.3.1 `ENTRANT* entrant_id_search ( int entrant_id, ENTRANT_LIST * entrants )`

Searches the entrant list for a entrant that has the passed ID.

#### Parameters

<i>entrant_id</i>	The ID of the entrant to search for.
<i>entrants</i>	List of entrants.

#### Returns

Returns a pointer to the entrant if found. If no entrant is found NULL is returned.

#### 4.6.3.2 `ENTRANT* entrant_name_search ( char * name, ENTRANT_LIST * entrants )`

Search the entrant list by name.

**Parameters**

<i>name</i>	Name of the entrant you are searching for.
<i>entrants</i>	List of entrants.

**Returns**

Returns a pointer to the entrant if found. If no entrant is found NULL is returned.

**4.6.3.3 void print\_entrant ( ENTRANT \* *entrant* )**

Prints out nicely formatted info about a entrant.

**Parameters**

<i>entrant</i>	The entrant to print out.
----------------	---------------------------

**4.6.3.4 ENTRANT\_LIST\* read\_entrants ( char \* *path* )**

This function opens a file containing the data about the entrants.

The function does some simple error checking on the file, and will return a negative integer on failure. The Data should include a positive integer that represents a id. A single capital character which is the id of the course the entrant has signed up for. Followed by a no longer than 80 characters long name.

**Parameters**

<i>path</i>	Path to the file you want to open.
-------------	------------------------------------

**Returns**

Returns 1 if successful, a negative integer otherwise.

## 4.7 event.c File Reference

This file contains function used to work with the event data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include "event.h"
#include "cp_time.h"
```

**Functions**

- **EVENT \* read\_event** (char \*path)

*This function opens a file containing the name of the event, the date and, the time.*

**Variables**

- int **errno**



### 4.7.1 Detailed Description

This file contains function used to work with the event data structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.12.02

### 4.7.2 Function Documentation

#### 4.7.2.1 `EVENT*` `read_event ( char * path )`

This function opens a file containing the name of the event, the date and, the time.

The file has 3 lines each no longer than 79 characters.

#### Parameters

<code>path</code>	Path to the file to read.
-------------------	---------------------------

#### Returns

Returns 1 if successful, a negative integer otherwise.

## 4.8 event.h File Reference

This file contains the definition of the event structure and other variables related to the event.

```
#include "entrant.h"
#include "node.h"
#include "track.h"
#include "course.h"
```

### Data Structures

- struct [event](#)

*The event structure holds information about the event such as the name, date, and time.*

### Typedefs

- typedef struct [event](#) [EVENT](#)

*The event structure holds information about the event such as the name, date, and time.*

### Functions

- [EVENT \\*](#) [read\\_event](#) (char \*path)

*This function opens a file containing the name of the event, the date and, the time.*

### 4.8.1 Detailed Description

This file contains the definition of the event structure and other variables related to the event.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.24

### 4.8.2 Typedef Documentation

#### 4.8.2.1 typedef struct event EVENT

The event structure holds information about the event such as the name, date, and time.

The structure also has a array of entrants and a linked list of the course nodes.

### 4.8.3 Function Documentation

#### 4.8.3.1 EVENT\* read\_event ( char \* path )

This function opens a file containing the name of the event, the date and, the time.

The file has 3 lines each no longer than 79 characters.

#### Parameters

<i>path</i>	Path to the file to read.
-------------	---------------------------

#### Returns

Returns 1 if successful, a negative integer otherwise.

## 4.9 functions.c File Reference

This file contains various functions for the program.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "functions.h"
#include "time_struct.h"
#include "entrant.h"
#include "event.h"
#include "course.h"
#include "cp_time.h"
#include "node.h"
#include "track.h"
```

#### Functions

- void [calc\\_time](#) (TIME\_STRUCT \*start, TIME\_STRUCT \*now, char \*total\_time)

- This function calculates the time the entrant has used so far.*
- `TIME_STRUCT * format_time (char *time_str)`  
*Extracts the time from a string, eg 07:30, and stores it in the TIME\_STRUCT type.*
- `void print_entrant_header ()`  
*Prints out the header used showing what each column is.*
- `void print_entrant (ENTRANT *entrant)`  
*Prints out nicely formatted info about a entrant.*
- `void not_started (EVENT *event)`  
*Prints out a nicely formatted list of entrants that has not started yet.*
- `void finished (EVENT *event)`  
*Prints out a nicely formatted list of entrants that has finished the course.*
- `void out_track (EVENT *event)`  
*Prints out a nicely formatted list of all entrants that are currently out on the track.*
- `void print_all (EVENT *event)`  
*Prints out a nicely formatted list of all entrants.*
- `int entrant_finished (EVENT *event, ENTRANT *entrant)`  
*This function checks if a user has finished the course the entrant is attending.*
- `void add_cp (EVENT *event, char status, NODE *node, ENTRANT *ent, char *time)`  
*This function is used to add a check point to a entrant.*

#### 4.9.1 Detailed Description

This file contains various functions for the program.

##### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

##### Date

2012.12.04

#### 4.9.2 Function Documentation

##### 4.9.2.1 void add\_cp ( EVENT \* event, char status, NODE \* node, ENTRANT \* ent, char \* time )

This function is used to add a check point to a entrant.

##### Parameters

<i>event</i>	Event structure with data about the event.
<i>status</i>	The status read from file.
<i>node</i>	Pointer to the node that is the checkpoint.
<i>ent</i>	The entrant that checked in at the check point
<i>time</i>	Time string of the time.

##### 4.9.2.2 void calc\_time ( TIME\_STRUCT \* start, TIME\_STRUCT \* now, char \* total\_time )

This function calculates the time the entrant has used so far.

## Parameters

<i>start</i>	The time the entrant started
<i>now</i>	The time now.
<i>total_time</i>	String buffer to store the result.

4.9.2.3 `int entrant_finished ( EVENT * event, ENTRANT * entrant )`

This function checks if a user has finished the course the entrant is attending.

The function creates a array of all check points in the course where time is recorded. It then compares the contents of the array with the checkpoints the entrant has been to. If they are in the same order and equal the entrant is finished.

## Parameters

<i>event</i>	Event structure containing data the function needs.
<i>entrant</i>	The entrant you want to check.

## Returns

If the entrant is finished 1 is returned, if the entrant is not finished 0 is returned.

4.9.2.4 `void finished ( EVENT * event )`

Prints out a nicely formatted list of entrants that has finished the course.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

4.9.2.5 `TIME_STRUCT* format_time ( char * time_str )`

Extracts the time from a string, eg 07:30, and stores it in the TIME\_STRUCT type.

## Parameters

<i>time_str</i>	The string containing the time you want to extract.
-----------------	---

## Returns

If successful the function returns a pointer to a TIME\_STRUCT. NULL is returned if the function fails.

4.9.2.6 `void not_started ( EVENT * event )`

Prints out a nicely formatted list of entrants that has not started yet.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

## 4.9.2.7 void out\_track ( EVENT \* event )

Prints out a nicely formatted list of all entrants that are currently out on the track.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

## 4.9.2.8 void print\_all ( EVENT \* event )

Prints out a nicely formatted list of all entrants.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

## 4.9.2.9 void print\_entrant ( ENTRANT \* entrant )

Prints out nicely formatted info about a entrant.

## Parameters

<i>entrant</i>	The entrant to print out.
----------------	---------------------------

## 4.10 functions.h File Reference

This file contains definitions of various functions.

```
#include "node.h"
#include "track.h"
#include "event.h"
#include "entrant.h"
#include "time_struct.h"
```

## Functions

- [TRACK\\_LIST \\* read\\_tracks](#) (char \*path, [NODE\\_LIST \\*list](#))  
*This function reads in and parses a file containing the track data.*
- [TIME\\_STRUCT \\* format\\_time](#) (char \*time\_str)  
*Extracts the time from a string, eg 07:30, and stores it in the TIME\_STRUCT type.*
- void [calc\\_time](#) ([TIME\\_STRUCT \\*start](#), [TIME\\_STRUCT \\*now](#), char \*total\_time)  
*This function calculates the time the entrant has used so far.*
- int [read\\_event\\_data](#) (char \*path, [EVENT \\*event](#))  
*This function reads the check point data from a file.*
- void [update\\_status](#) ([ENTRANT \\*ent](#), [EVENT \\*event](#))  
*Updates the status of entrant.*
- void [not\\_started](#) ([EVENT \\*event](#))  
*Prints out a nicely formatted list of entrants that has not started yet.*
- void [finished](#) ([EVENT \\*event](#))  
*Prints out a nicely formatted list of entrants that has finished the course.*
- void [out\\_track](#) ([EVENT \\*event](#))

*Prints out a nicely formatted list of all entrants that are currently out on the track.*

- void `print_all` (`EVENT *event`)

*Prints out a nicely formatted list of all entrants.*

- int `entrant_finished` (`EVENT *event`, `ENTRANT *entrant`)

*This function checks if a user has finished the course the entrant is attending.*

- void `add_cp` (`EVENT *event`, char `status`, `NODE *node`, `ENTRANT *ent`, char `*time`)

*This function is used to add a check point to a entrant.*

#### 4.10.1 Detailed Description

This file contains definitions of various functions.

##### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

##### Date

2012.11.30

#### 4.10.2 Function Documentation

##### 4.10.2.1 void `add_cp` ( `EVENT *event`, char `status`, `NODE *node`, `ENTRANT *ent`, char `*time` )

This function is used to add a check point to a entrant.

##### Parameters

<i>event</i>	Event structure with data about the event.
<i>status</i>	The status read from file.
<i>node</i>	Pointer to the node that is the checkpoint.
<i>ent</i>	The entrant that checked in at the check point
<i>time</i>	Time string of the time.

##### 4.10.2.2 void `calc_time` ( `TIME_STRUCT *start`, `TIME_STRUCT *now`, char `*total_time` )

This function calculates the time the entrant has used so far.

##### Parameters

<i>start</i>	The time the entrant started
<i>now</i>	The time now.
<i>total_time</i>	String buffer to store the result.

##### 4.10.2.3 int `entrant_finished` ( `EVENT *event`, `ENTRANT *entrant` )

This function checks if a user has finished the course the entrant is attending.

The function creates a array of all check points in the course where time is recorded. It then compares the contents of the array with the checkpoints the entrant has been to. If they are in the same order and equal the entrant is finished.

## Parameters

<i>event</i>	Event structure containing data the function needs.
<i>entrant</i>	The entrant you want to check.

## Returns

If the entrant is finished 1 is returned, if the entrant is not finished 0 is returned.

4.10.2.4 void finished ( EVENT \* *event* )

Prints out a nicely formatted list of entrants that has finished the course.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

4.10.2.5 TIME\_STRUCT\* format\_time ( char \* *time\_str* )

Extracts the time from a string, eg 07:30, and stores it in the TIME\_STRUCT type.

## Parameters

<i>time_str</i>	The string containing the time you want to extract.
-----------------	---

## Returns

If successful the function returns a pointer to a TIME\_STRUCT. NULL is returned if the function fails.

4.10.2.6 void not\_started ( EVENT \* *event* )

Prints out a nicely formatted list of entrants that has not started yet.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

4.10.2.7 void out\_track ( EVENT \* *event* )

Prints out a nicely formatted list of all entrants that are currently out on the track.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

4.10.2.8 void print\_all ( EVENT \* *event* )

Prints out a nicely formatted list of all entrants.

## Parameters

<i>event</i>	Event structure.
--------------	------------------

#### 4.10.2.9 `int read_event_data ( char * path, EVENT * event )`

This function reads the check point data from a file.

##### Parameters

<i>path</i>	Path to the file containing the checkpoint data.
<i>event</i>	Event structure.

##### Returns

On success this function returns 1, 0 is returned on failure.

#### 4.10.2.10 `TRACK_LIST* read_tracks ( char * path, NODE_LIST * node_list )`

This function reads in and parses a file containing the track data.

It uses this data to create a linked list of the tracks. The file should have data about paths between nodes and the maximum time a entrant should use between the nodes.

##### Parameters

<i>path</i>	The path to the file containing the track data.
<i>node_list</i>	Pointer to the list of nodes, the list is needed to look up the nodes that are used in the track data to verify that the track is between nodes that exist.

##### Returns

Upon success this function returns a pointer to the first element in the linked list. If it fails it will return NULL.

#### 4.10.2.11 `void update_status ( ENTRANT * ent, EVENT * event )`

Updates the status of entrant.

Valid statues are "Not Started", "Finished", or the id of the last node the entrant checked in at.

##### Parameters

<i>ent</i>	The entrant to update.
<i>event</i>	Event structure.

## 4.11 input.c File Reference

This file holds functions related to getting input from the user.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "entrant.h"
#include "node.h"
#include "track.h"
#include "course.h"
#include "functions.h"
```



## Functions

- `int ask_int (char *question, int min, int max)`  
*This function asks the user a question and takes a single integer between a set range as input.*
- `void ask_str (char *question, char *response)`  
*This function ask the user a question and takes a line or up to 255 characters of input.*
- `ENTRANT_LIST * get_entrants ()`  
*This function prompts the user with a question asking for the path to a file containing data on entrants.*
- `NODE_LIST * get_nodes ()`  
*This function prompts the user with a question asking for the path to a file containing data on nodes.*
- `TRACK_LIST * get_tracks (NODE_LIST *node_list)`  
*This function prompts the user with a question asking for the path to a file containing data on tracks.*
- `EVENT * get_event ()`  
*Prompts the user with a question about the path to the file containing the date about the event.*
- `COURSE_LIST * get_courses (NODE_LIST *nodes, TRACK_LIST *tracks)`  
*Prompts the user about the path to a file containing data about courses.*
- `void get_cp_data (EVENT *event)`  
*Prompts the user with the question to enter the path to the filename containing the checkpoint data.*
- `void load_cp_data (EVENT *event)`  
*Prompts the user with the choice of loading the check point data from file or entering it manually.*
- `int show_menu ()`  
*This function prints out the main user menu the user will use to interact with the different parts of the program .*
- `ENTRANT * query_entrant (EVENT *event)`  
*Searches for a entrant either by name or id given by the user.*

### 4.11.1 Detailed Description

This file holds functions related to getting input from the user.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.12.1

### 4.11.2 Function Documentation

#### 4.11.2.1 `int ask_int ( char * question, int min, int max )`

This function asks the user a question and takes a single integer between a set range as input.

The function will repeat the question until a valid integer is entered as input.

#### Parameters

<i>question</i>	String with the question you want to ask the user.
<i>min</i>	Minimum limit of the range.
<i>max</i>	Maximum limit of the range.

**Returns**

The integer value the user inputted.

**4.11.2.2 void ask\_str ( char \* *question*, char \* *response* )**

This function ask the user a question and takes a line or up to 255 characters of input.

The input is copied into a buffer which should not be smaller than 256 characters.

**Parameters**

<i>question</i>	String with the question you want to ask the user.
<i>response</i>	The buffer the response is copied into. This buffer should not be smaller than 256 characters.

**4.11.2.3 COURSE\_LIST\* get\_courses ( NODE\_LIST \* *nodes*, TRACK\_LIST \* *tracks* )**

Prompts the user about the path to a file containing data about courses.

The data should consist of a single capital character which will be the course ID. Followed by a positive integer which will be the number of nodes for the course. The rest of the line should be the ID of each node separated by spaces.

**Parameters**

<i>nodes</i>	Pointer to a populated linked list of nodes. The nodes are needed to verify that the course is made of known nodes.
<i>tracks</i>	Pointer to a populated linked list of tracks. The tracks are needed to verify that there is a valid path from start to goal.

**Returns**

The function returns a pointer to a linked list of courses if successful. If it fails NULL is returned.

**4.11.2.4 void get\_cp\_data ( EVENT \* *event* )**

Prompts the user with the question to enter the path to the filename containing the checkpoint data.

**Parameters**

<i>event</i>	The result are stored in this variable.
--------------	---

**4.11.2.5 ENTRANT\_LIST\* get\_entrants ( )**

This function prompts the user with a question asking for the path to a file containing data on entrants.

The Data should include a positive integer that represents a id. A single capital character which is the id of the course the entrant has signed up for. Followed by a no longer than 80 characters long name.

**Returns**

The function returns a pointer to a ENTRANT array if the function was successful. If it fails the function returns NULL.

**4.11.2.6 EVENT\* get\_event ( )**

Prompts the user with a question about the path to the file containing the data about the event.

The file should have data about the name event, start time and date.

**Returns**

If the function is successful it will return a pointer to a event populated with the data from the file. If it fails NULL is returned.

**4.11.2.7 NODE\_LIST\* get\_nodes ( )**

This function prompts the user with a question asking for the path to a file containing data on nodes.

The data should contain a node ID and a short string used to identify what type of node the node is.

**Returns**

The function returns a pointer to a NODE\_LIST, the NODE\_LIST will be populated with data if successful. If it fails NULL is returned.

**4.11.2.8 TRACK\_LIST\* get\_tracks ( NODE\_LIST \* node\_list )**

This function prompts the user with a question asking for the path to a file containing data on tracks.

The file should have data about paths between nodes and the maximum time a entrant should use between the nodes.

**Parameters**

<i>node_list</i>	A pointer to a NODE_LIST, it should be populated. The nodes are needed to ensure that the tracks does not contain none existing nodes.
------------------	--

**Returns**

This function returns a pointer to a TRACK\_LIST if successful. If the function fails NULL is returned.

**4.11.2.9 void load\_cp\_data ( EVENT \* event )**

Prompts the user with the choice of loading the check point data from file or entering it manually.

**Parameters**

<i>event</i>	Event data structure. The result will be stored in this variable.
--------------	---

**4.11.2.10 ENTRANT\* query\_entrant ( EVENT \* event )**

Searches for a entrant either by name or id given by the user.

The function prompts the user with the choice of searching by name or by the entrant id.

**Parameters**

<i>event</i>	Event structure. The result will be stored in this variable.
--------------	--

**Returns**

If successful the function returns a pointer to the entrant. If it fails NULL is returned.

**4.11.2.11 int show\_menu ( )**

This function prints out the main user menu the user will use to interact with the different parts of the program .

The user is given the choice to go to several different sub menus. The function takes a integer as input and returns the input.

**Returns**

User inputted integer.

**4.12 input.h File Reference**

This file contains function definitions for [input.c](#).

```
#include "entrant.h"
```

**Functions**

- [ENTRANT\\_LIST](#) \* [get\\_entrants](#) ()  
*This function prompts the user with a question asking for the path to a file containing data on entrants.*
- [NODE\\_LIST](#) \* [get\\_nodes](#) ()  
*This function prompts the user with a question asking for the path to a file containing data on nodes.*
- [TRACK\\_LIST](#) \* [get\\_tracks](#) ([NODE\\_LIST](#) \*[node\\_list](#))  
*This function prompts the user with a question asking for the path to a file containing data on tracks.*
- [EVENT](#) \* [get\\_event](#) ()  
*Prompts the user with a question about the path to the file containing the date about the event.*
- [COURSE\\_LIST](#) \* [get\\_courses](#) ([NODE\\_LIST](#) \*[nodes](#), [TRACK\\_LIST](#) \*[tracks](#))  
*Prompts the user about the path to a file containing data about courses.*
- int [show\\_menu](#) ()  
*This function prints out the main user menu the user will use to interact with the different parts of the program .*
- [ENTRANT](#) \* [query\\_entrant](#) ([EVENT](#) \*[event](#))  
*Searches for a entrant either by name or id given by the user.*
- int [ask\\_int](#) (char \*[question](#), int min, int max)  
*This function asks the user a question and takes a single integer between a set range as input.*
- void [ask\\_str](#) (char \*[question](#), char \*[response](#))  
*This function ask the user a question and takes a line or up to 255 characters of input.*
- void [get\\_cp\\_data](#) ([EVENT](#) \*[event](#))  
*Prompts the user with the question to enter the path to the filename containing the checkpoint data.*
- void [load\\_cp\\_data](#) ([EVENT](#) \*[event](#))  
*Prompts the user with the choice of loading the check point data from file or entering it manually.*

### 4.12.1 Detailed Description

This file contains function definitions for [input.c](#).

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.30

### 4.12.2 Function Documentation

#### 4.12.2.1 `int ask_int ( char * question, int min, int max )`

This function asks the user a question and takes a single integer between a set range as input.

The function will repeat the question until a valid integer is entered as input.

##### Parameters

<i>question</i>	String with the question you want to ask the user.
<i>min</i>	Minimum limit of the range.
<i>max</i>	Maximum limit of the range.

##### Returns

The integer value the user inputted.

#### 4.12.2.2 `void ask_str ( char * question, char * response )`

This function ask the user a question and takes a line or up to 255 characters of input.

The input is copied into a buffer which should not be smaller than 256 characters.

##### Parameters

<i>question</i>	String with the question you want to ask the user.
<i>response</i>	The buffer the response is copied into. This buffer should not be smaller than 256 characters.

#### 4.12.2.3 `COURSE_LIST* get_courses ( NODE_LIST * nodes, TRACK_LIST * tracks )`

Prompts the user about the path to a file containing data about courses.

The data should consist of a single capital character which will be the course ID. Followed by a positive integer which will be the number of nodes for the course. The rest of the line should be the ID of each node separated by spaces.

##### Parameters

<i>nodes</i>	Pointer to a populated linked list of nodes. The nodes are needed to verify that the course is made of known nodes.
<i>tracks</i>	Pointer to a populated linked list of tracks. The tracks are needed to verify that there is a valid path from start to goal.

**Returns**

The function returns a pointer to a linked list of courses if successful. If it fails NULL is returned.

**4.12.2.4 void get\_cp\_data ( EVENT \* event )**

Prompts the user with the question to enter the path to the filename containing the checkpoint data.

**Parameters**

<i>event</i>	The result are stored in this variable.
--------------	---

**4.12.2.5 ENTRANT\_LIST\* get\_entrants ( )**

This function prompts the user with a question asking for the path to a file containing data on entrants.

The Data should include a positive integer that represents a id. A single capital character which is the id of the course the entrant has signed up for. Followed by a no longer than 80 characters long name.

**Returns**

The function returns a pointer to a ENTRANT array if the function was successful. If it fails the function returns NULL.

**4.12.2.6 EVENT\* get\_event ( )**

Prompts the user with a question about the path to the file containing the date about the event.

The file should have date about the name event, start time and date.

**Returns**

If the function is successful it will return a pointer to a event populated with the data from the file. If it fails NULL is returned.

**4.12.2.7 NODE\_LIST\* get\_nodes ( )**

This function prompts the user with a question asking for the path to a file containing data on nodes.

The data should contain a node ID and a short string used to identify what type of node the node is.

**Returns**

The function returns a pointer to a NODE\_LIST, the NODE\_LIST will be populated with data if successful. If it fails NULL is returned.

**4.12.2.8 TRACK\_LIST\* get\_tracks ( NODE\_LIST \* node\_list )**

This function prompts the user with a question asking for the path to a file containing data on tracks.

The file should have data about paths between nodes and the maximum time a entrant should use between the nodes.

**Parameters**

<i>node_list</i>	A pointer to a NODE_LIST, it should be populated. The nodes are needed to ensure that the tracks does not contain none existing nodes.
------------------	--

**Returns**

This function returns a pointer to a TRACK\_LIST if successful. If the function fails NULL is returned.

**4.12.2.9 void load\_cp\_data ( EVENT \* event )**

Prompts the user with the choice of loading the check point data from file or entering it manually.

**Parameters**

<i>event</i>	Event data structure. The result will be stored in this variable.
--------------	---

**4.12.2.10 ENTRANT\* query\_entrant ( EVENT \* event )**

Searches for a entrant either by name or id given by the user.

The function prompts the user with the choice of searching by name or by the entrant id.

**Parameters**

<i>event</i>	Event structure. The result will be stored in this variable.
--------------	--

**Returns**

If successful the function returns a pointer to the entrant. If it fails NULL is returned.

**4.12.2.11 int show\_menu ( )**

This function prints out the main user menu the user will use to interact with the different parts of the program .

The user is given the choice to go to several different sub menus. The function takes a integer as input and returns the input.

**Returns**

User inputed integer.

## 4.13 node.c File Reference

This file holds function used to wok with the node data structure.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include "node.h"
```

**Functions**

- **NODE \* node\_id\_search** (int id, **NODE\_LIST** \*list)  
*This function searches for a node with a given ID.*
- **NODE\_LIST \* read\_nodes** (char \*path)  
*This functions read in nodes from file and creates a linked list of the nodes.*

## Variables

- int **errno**

### 4.13.1 Detailed Description

This file holds function used to wok with the node data structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.25

### 4.13.2 Function Documentation

#### 4.13.2.1 **NODE\*** node\_id\_search ( int *id*, **NODE\_LIST** \* *list* )

This function searches for a node with a given ID.

##### Parameters

<i>id</i>	The ID of the node you want to search for.
<i>list</i>	Pointer to the list, this list is the list that will be searched.

##### Returns

If the node is found the function returns the a pointer to the node. If the node is not found it will return NULL.

#### 4.13.2.2 **NODE\_LIST\*** read\_nodes ( char \* *path* )

This functions read in nodes from file and creates a linked list of the nodes.

The data should contain a node ID and a short string used to identify what type of node the node is.

##### Parameters

<i>path</i>	Path to the file where the node data is stored.
-------------	---

##### Returns

Returns a pointer to the first NODE in the linked list. Returns NULL if something went wrong.

Need to set a start if it hasn't been done yet.

## 4.14 node.h File Reference

This file contains the definition of the node data structure and definitions of functions used to work with the data structure.

```
#include "track.h"
```



## Data Structures

- struct `node`

*The node data structure represents a checkpoint in the course.*

- struct `node_list`

*This is the node list.*

## Typedefs

- typedef struct `node` `NODE`

*The node data structure represents a checkpoint in the course.*

- typedef struct `node_list` `NODE_LIST`

*This is the node list.*

## Functions

- `NODE * node_id_search` (int id, `NODE_LIST *list`)

*This function searches for a node with a given ID.*

- `NODE_LIST * read_nodes` (char \*path)

*This functions read in nodes from file and creates a linked list of the nodes.*

### 4.14.1 Detailed Description

This file contains the definition of the node data structure and definitions of functions used to work with the data structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.25

### 4.14.2 Typedef Documentation

#### 4.14.2.1 typedef struct `node` `NODE`

The node data structure represents a checkpoint in the course.

There are several different checkpoints. Each node is identified by a positive integer. The data structure is also a linked list and has a pointer to the next element in the list.

#### 4.14.2.2 typedef struct `node_list` `NODE_LIST`

This is the node list.

It is used to easily pass around the list and information about it to make working with it easier.

### 4.14.3 Function Documentation

#### 4.14.3.1 **NODE\*** node\_id\_search ( int *id*, **NODE\_LIST** \* *list* )

This function searches for a node with a given ID.

##### Parameters

<i>id</i>	The ID of the node you want to search for.
<i>list</i>	Pointer to the list, this list is the list that will be searched.

##### Returns

If the node is found the function returns the a pointer to the node. If the node is not found it will return NULL.

#### 4.14.3.2 **NODE\_LIST\*** read\_nodes ( char \* *path* )

This functions read in nodes from file and creates a linked list of the nodes.

The data should contain a node ID and a short string used to identify what type of node the node is.

##### Parameters

<i>path</i>	Path to the file where the node data is stored.
-------------	---

##### Returns

Returns a pointer to the first NODE in the linked list. Returns NULL if something went wrong.

Need to set a start if it hasn't been done yet.

## 4.15 tests.h File Reference

This file contains the definitions of several functi.

### Functions

- void [entrant\\_read](#) ()  
*Test reading and parsing of entrant file.*
- void **name\_read** ()
- void **node\_read** ()
- void **track\_read** ()
- void [course\\_read](#) ()  
  

```
void track_read() { NODE *start = read_nodes("data/nodes.txt"); TRACK *track_start = read_tracks("data/tracks.txt", start);
```
- void **event\_data\_read** ()
- void **test\_calc\_time** ()
- void **test\_all\_files** ()
- void **entrant\_search\_test** ()
- void **test** ()

### 4.15.1 Detailed Description

This file contains the definitions of several functi.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

on 24 November 2012, 02:09

### 4.15.2 Function Documentation

#### 4.15.2.1 void course\_read ( )

```
void track_read() { NODE *start = read_nodes("data/nodes.txt"); TRACK *track_start = read_tracks("data/tracks.-txt", start);
```

```
printf("breakpoint\n");
```

```
}
```

## 4.16 time\_struct.h File Reference

This file contains the definition of the data structure used to represent the time structure.

### Data Structures

- struct [time\\_struct](#)

*The structure used to represent time.*

### Typedefs

- typedef struct [time\\_struct](#) TIME\_STRUCT

*The structure used to represent time.*

### 4.16.1 Detailed Description

This file contains the definition of the data structure used to represent the time structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.12.07

## 4.17 track.c File Reference

This file contains functions used to work with the track data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include "track.h"
#include "node.h"
```

### Functions

- [TRACK\\_LIST](#) \* [read\\_tracks](#) (char \*path, [NODE\\_LIST](#) \*node\_list)  
*This function reads in and parses a file containing the track data.*
- int [validate\\_track](#) (int node\_one, int node\_two, [TRACK](#) track)  
*Checks if the given track is between the given nodes.*
- void [free\\_track](#) ([TRACK\\_LIST](#) \*track\_list)

### Variables

- int [errno](#)

### 4.17.1 Detailed Description

This file contains functions used to work with the track data structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.30

### 4.17.2 Function Documentation

#### 4.17.2.1 [TRACK\\_LIST](#)\* [read\\_tracks](#) ( char \* path, [NODE\\_LIST](#) \* node\_list )

This function reads in and parses a file containing the track data.

It uses this data to create a linked list of the tracks. The file should have data about paths between nodes and the maximum time a entrant should use between the nodes.

#### Parameters

<a href="#">path</a>	The path to the file containing the track data.
<a href="#">node_list</a>	Pointer to the list of nodes, the list is needed to look up the nodes that are used in the track data to verify that the track is between nodes that exist.

#### Returns

Upon success this function returns a pointer to the first element in the linked list. If it fails it will return NULL.

4.17.2.2 `int validate_track ( int node_one, int node_two, TRACK track )`

Checks if the given track is between the given nodes.

#### Parameters

<code>node_one</code>	ID of one of the NODEs.
<code>node_two</code>	The ID of the other NODE.
<code>track</code>	The TRACK you want to check.

#### Returns

Returns 1 if the TRACK is between NODEs. If fails 0 is returned.

## 4.18 track.h File Reference

This file contains the definition of the track structure.

### Data Structures

- struct `track`  
*This structure holds the information about each track.*
- struct `track_list`  
*This is the track list.*

### Typedefs

- typedef struct `track` `TRACK`  
*This structure holds the information about each track.*
- typedef struct `track_list` `TRACK_LIST`  
*This is the track list.*

### Functions

- int `validate_track` (int node\_one, int node\_two, `TRACK` track)  
*Checks if the given track is between the given nodes.*
- void `free_track` (`TRACK_LIST` \*`track_list`)

#### 4.18.1 Detailed Description

This file contains the definition of the track structure.

#### Author

Sindre Smistad [sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)

#### Date

2012.11.24

## 4.18.2 Typedef Documentation

### 4.18.2.1 typedef struct track TRACK

This structure holds the information about each track.

The structure also has a pointer to the another track to act as a linked list.

### 4.18.2.2 typedef struct track\_list TRACK\_LIST

This is the track list.

It is used to easily pass around the list and information about it to make working with it easier.

## 4.18.3 Function Documentation

### 4.18.3.1 int validate\_track ( int *node\_one*, int *node\_two*, TRACK *track* )

Checks if the given track is between the given nodes.

#### Parameters

<i>node_one</i>	ID of one of the NODEs.
<i>node_two</i>	The ID of the other NODE.
<i>track</i>	The TRACK you want to check.

#### Returns

Returns 1 if the TRACK is between NODEs. If fails 0 is returned.

# Index

- add\_cp
  - functions.c, [27](#)
  - functions.h, [30](#)
- ask\_int
  - input.c, [33](#)
  - input.h, [37](#)
- ask\_str
  - input.c, [34](#)
  - input.h, [37](#)
- COURSE
  - course.h, [17](#)
- COURSE\_LIST
  - course.h, [17](#)
- calc\_time
  - functions.c, [27](#)
  - functions.h, [30](#)
- course, [5](#)
  - course\_id, [5](#)
- course.c, [15](#)
  - read\_courses, [16](#)
  - search\_course\_id, [16](#)
  - validate\_course, [16](#)
- course.h, [16](#)
  - COURSE, [17](#)
  - COURSE\_LIST, [17](#)
  - read\_courses, [18](#)
  - search\_course\_id, [18](#)
  - validate\_course, [18](#)
- course\_id
  - course, [5](#)
- course\_list, [6](#)
  - entrants, [6](#)
  - node\_list, [6](#)
  - track\_list, [6](#)
- course\_read
  - tests.h, [43](#)
- cp\_time, [7](#)
- cp\_time.c, [19](#)
  - read\_event\_data, [19](#)
- cp\_time.h, [19](#)
- cp\_time\_list, [7](#)
- date
  - event, [10](#)
- ENTRANT\_LIST
  - entrant.h, [23](#)
- EVENT
  - event.h, [26](#)
- entrant, [7](#)
  - status, [8](#)
- entrant.c, [20](#)
  - entrant\_id\_search, [21](#)
  - entrant\_name\_search, [21](#)
  - read\_entrants, [21](#)
  - update\_status, [22](#)
- entrant.h, [22](#)
  - ENTRANT\_LIST, [23](#)
  - entrant\_id\_search, [23](#)
  - entrant\_name\_search, [23](#)
  - print\_entrant, [24](#)
  - read\_entrants, [24](#)
- entrant\_finished
  - functions.c, [28](#)
  - functions.h, [30](#)
- entrant\_id\_search
  - entrant.c, [21](#)
  - entrant.h, [23](#)
- entrant\_list, [8](#)
- entrant\_name\_search
  - entrant.c, [21](#)
  - entrant.h, [23](#)
- entrants
  - course\_list, [6](#)
  - event, [10](#)
- event, [9](#)
  - date, [10](#)
  - entrants, [10](#)
  - name, [10](#)
- event.c, [24](#)
  - read\_event, [25](#)
- event.h, [25](#)
  - EVENT, [26](#)
  - read\_event, [26](#)
- finished
  - functions.c, [28](#)
  - functions.h, [31](#)
- format\_time
  - functions.c, [28](#)
  - functions.h, [31](#)
- functions.c, [26](#)
  - add\_cp, [27](#)
  - calc\_time, [27](#)
  - entrant\_finished, [28](#)
  - finished, [28](#)
  - format\_time, [28](#)
  - not\_started, [28](#)
  - out\_track, [28](#)

- print\_all, 29
- print\_entrant, 29
- functions.h, 29
  - add\_cp, 30
  - calc\_time, 30
  - entrant\_finished, 30
  - finished, 31
  - format\_time, 31
  - not\_started, 31
  - out\_track, 31
  - print\_all, 31
  - read\_event\_data, 31
  - read\_tracks, 32
  - update\_status, 32
- get\_courses
  - input.c, 34
  - input.h, 37
- get\_cp\_data
  - input.c, 34
  - input.h, 38
- get\_entrants
  - input.c, 34
  - input.h, 38
- get\_event
  - input.c, 34
  - input.h, 38
- get\_nodes
  - input.c, 35
  - input.h, 38
- get\_tracks
  - input.c, 35
  - input.h, 38
- input.c, 32
  - ask\_int, 33
  - ask\_str, 34
  - get\_courses, 34
  - get\_cp\_data, 34
  - get\_entrants, 34
  - get\_event, 34
  - get\_nodes, 35
  - get\_tracks, 35
  - load\_cp\_data, 35
  - query\_entrant, 35
  - show\_menu, 36
- input.h, 36
  - ask\_int, 37
  - ask\_str, 37
  - get\_courses, 37
  - get\_cp\_data, 38
  - get\_entrants, 38
  - get\_event, 38
  - get\_nodes, 38
  - get\_tracks, 38
  - load\_cp\_data, 39
  - query\_entrant, 39
  - show\_menu, 39
- load\_cp\_data
  - input.c, 35
  - input.h, 39
- NODE
  - node.h, 41
- NODE\_LIST
  - node.h, 41
- name
  - event, 10
- node, 10
  - node\_id, 11
- node.c, 39
  - node\_id\_search, 40
  - read\_nodes, 40
- node.h, 40
  - NODE, 41
  - NODE\_LIST, 41
  - node\_id\_search, 42
  - read\_nodes, 42
- node\_id
  - node, 11
- node\_id\_search
  - node.c, 40
  - node.h, 42
- node\_list, 11
  - course\_list, 6
- not\_started
  - functions.c, 28
  - functions.h, 31
- out\_track
  - functions.c, 28
  - functions.h, 31
- print\_all
  - functions.c, 29
  - functions.h, 31
- print\_entrant
  - entrant.h, 24
  - functions.c, 29
- query\_entrant
  - input.c, 35
  - input.h, 39
- read\_courses
  - course.c, 16
  - course.h, 18
- read\_entrants
  - entrant.c, 21
  - entrant.h, 24
- read\_event
  - event.c, 25
  - event.h, 26
- read\_event\_data
  - cp\_time.c, 19
  - functions.h, 31
- read\_nodes



- node.c, [40](#)
  - node.h, [42](#)
- read\_tracks
  - functions.h, [32](#)
  - track.c, [44](#)
- search\_course\_id
  - course.c, [16](#)
  - course.h, [18](#)
- show\_menu
  - input.c, [36](#)
  - input.h, [39](#)
- status
  - entrant, [8](#)
- TRACK
  - track.h, [46](#)
- TRACK\_LIST
  - track.h, [46](#)
- tests.h, [42](#)
  - course\_read, [43](#)
- time\_struct, [11](#)
- time\_struct.h, [43](#)
- track, [12](#)
- track.c, [44](#)
  - read\_tracks, [44](#)
  - validate\_track, [44](#)
- track.h, [45](#)
  - TRACK, [46](#)
  - TRACK\_LIST, [46](#)
  - validate\_track, [46](#)
- track\_list, [12](#)
  - course\_list, [6](#)
- update\_status
  - entrant.c, [22](#)
  - functions.h, [32](#)
- validate\_course
  - course.c, [16](#)
  - course.h, [18](#)
- validate\_track
  - track.c, [44](#)
  - track.h, [46](#)