

CS23710 Assignment

Generated by Doxygen 1.8.1.2

Fri Dec 14 2012 10:35:12

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
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	8
3.5.1	Detailed Description	8
3.5.2	Field Documentation	8
3.5.2.1	excluded	8
3.5.2.2	mc_excluded	8
3.5.2.3	status	9
3.6	entrant_list Struct Reference	9
3.6.1	Detailed Description	9
3.7	event Struct Reference	9
3.7.1	Detailed Description	10
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	11
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
4	File Documentation	15
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	entrant_name_search	21
4.5.2.3	read_entrants	21
4.5.2.4	update_status	22
4.6	entrant.h File Reference	22
4.6.1	Detailed Description	23
4.6.2	Typedef Documentation	23
4.6.2.1	ENTRANT_LIST	23
4.6.3	Function Documentation	23
4.6.3.1	entrant_id_search	23
4.6.3.2	entrant_name_search	23
4.6.3.3	print_entrant	24
4.6.3.4	print_entrant_header	24
4.6.3.5	read_entrants	24
4.7	event.c File Reference	24
4.7.1	Detailed Description	25
4.7.2	Function Documentation	25
4.7.2.1	read_event	25
4.8	event.h File Reference	25
4.8.1	Detailed Description	26
4.8.2	Typedef Documentation	26
4.8.2.1	EVENT	26
4.8.3	Function Documentation	26
4.8.3.1	read_event	26
4.9	functions.c File Reference	26
4.9.1	Detailed Description	27
4.9.2	Function Documentation	27
4.9.2.1	add_cp	27
4.9.2.2	add_mc	28
4.9.2.3	calc_time	28
4.9.2.4	entrant_finished	28
4.9.2.5	finished	28
4.9.2.6	format_time	29
4.9.2.7	not_started	29
4.9.2.8	out_track	29
4.9.2.9	print_all	29
4.9.2.10	print_entrant	29
4.9.2.11	print_entrant_header	29
4.9.2.12	print_excluded	30

4.10 functions.h File Reference	30
4.10.1 Detailed Description	31
4.10.2 Function Documentation	31
4.10.2.1 add_cp	31
4.10.2.2 add_mc	31
4.10.2.3 calc_time	31
4.10.2.4 entrant_finished	32
4.10.2.5 finished	32
4.10.2.6 format_time	32
4.10.2.7 not_started	32
4.10.2.8 out_track	32
4.10.2.9 print_all	33
4.10.2.10 print_excluded	33
4.10.2.11 read_event_data	33
4.10.2.12 read_tracks	33
4.10.2.13 update_status	33
4.11 input.c File Reference	34
4.11.1 Detailed Description	34
4.11.2 Function Documentation	35
4.11.2.1 ask_int	35
4.11.2.2 ask_str	35
4.11.2.3 get_courses	35
4.11.2.4 get_cp_data	35
4.11.2.5 get_entrants	36
4.11.2.6 get_event	36
4.11.2.7 get_nodes	36
4.11.2.8 get_tracks	36
4.11.2.9 load_cp_data	36
4.11.2.10 query_entrant	37
4.11.2.11 show_menu	37
4.12 input.h File Reference	37
4.12.1 Detailed Description	38
4.12.2 Function Documentation	38
4.12.2.1 ask_int	38
4.12.2.2 ask_str	38
4.12.2.3 get_courses	38
4.12.2.4 get_cp_data	39
4.12.2.5 get_entrants	39
4.12.2.6 get_event	39
4.12.2.7 get_nodes	39

4.12.2.8	get_tracks	40
4.12.2.9	load_cp_data	40
4.12.2.10	query_entrant	40
4.12.2.11	show_menu	40
4.13	main.c File Reference	40
4.13.1	Detailed Description	41
4.14	node.c File Reference	41
4.14.1	Detailed Description	41
4.14.2	Function Documentation	42
4.14.2.1	node_id_search	42
4.14.2.2	read_nodes	42
4.15	node.h File Reference	42
4.15.1	Detailed Description	43
4.15.2	Typedef Documentation	43
4.15.2.1	NODE	43
4.15.2.2	NODE_LIST	43
4.15.3	Function Documentation	43
4.15.3.1	node_id_search	43
4.15.3.2	read_nodes	44
4.16	tests.h File Reference	44
4.16.1	Detailed Description	44
4.16.2	Function Documentation	45
4.16.2.1	course_read	45
4.17	time_struct.h File Reference	45
4.17.1	Detailed Description	45
4.18	track.c File Reference	45
4.18.1	Detailed Description	46
4.18.2	Function Documentation	46
4.18.2.1	read_tracks	46
4.18.2.2	validate_track	46
4.19	track.h File Reference	47
4.19.1	Detailed Description	47
4.19.2	Typedef Documentation	47
4.19.2.1	TRACK	47
4.19.2.2	TRACK_LIST	47
4.19.3	Function Documentation	48
4.19.3.1	validate_track	48

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

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

course.c	This file contains functions used to work with the course data structure	15
course.h	This file contains the definition of the course data structure and definitions of functions used to work with the data structure	16
cp_time.c	This file contains functions used to work with the CP_TIME data structure	19
cp_time.h	This file contains the definition of the data structure used to represent a entrant checking in at a check point	19
entrant.c	This file holds the functions used to work with the ENTRANT data structure	20
entrant.h	This file holds structures and values needed to work with entrants	22
event.c	This file contains function used to work with the event data structure	24
event.h	This file contains the definition of the event structure and other variables related to the event	25
functions.c	This file contains various functions for the program	26
functions.h	This file contains definitions of various functions	30
input.c	This file holds functions related to getting input from the user	34
input.h	This file contains function definitions for input.c	37
main.c	40
node.c	This file holds function used to wok with the node data structure	41
node.h	This file contains the definition of the node data structure and definitions of functions used to work with the data structure	42
tests.h	This file contains the definitions of several functi	44
time_struct.h	This file contains the definition of the data structure used to represent the time structure	45
track.c	This file contains functions used to work with the track data structure	45

[track.h](#)

This file contains the definition of the track structure [47](#)

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.
- int [medical](#)
Is set if this check point is a medical check point.
- [TIME_STRUCT](#) * [time](#)
The string containing the time the entrant reached the CP.
- [TIME_STRUCT](#) * [departure](#)
The departure time from the medical check point.
- 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.
- int `mc_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 int entrant::excluded

Boolean used to denote if the entrant has been excluded from the race.

This is set if the entrant has been excluded for taking the wrong turn.

3.5.2.2 int entrant::mc_excluded

Boolean used to denote if the entrant has been excluded from the race.

This is set if the entrant has been excluded for failing a medical check point.

3.5.2.3 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 pinter 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

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

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

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

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%-50s%-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

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 void print_entrant_header ()

Prints out the header used showing what each column is.

ID Name Course Status Start time Total time

4.6.3.5 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

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

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` (`ENTRANT *ent`)
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.
- void `print_excluded` (`EVENT *event`, `int mc`)
Prints out the a list of entrants that has been excluded from the race.
- 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.
- void `add_mc` (`EVENT *event`, `char status`, `NODE *node`, `ENTRANT *ent`, `char *time`)
This function is used to add a medical 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

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.

The function will create a check point list in the entrant if this is the entrant's first check point. If the check point status is 'I' the entrant will be marked as excluded by this function. Total time of the entrant will also be updated. For adding medical checkpoints see `add_mc`.

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 add_mc (EVENT * event, char status, NODE * node, ENTRANT * ent, char * time)

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

The function will create a check point list in the entrant if this is the entrant's first check point. If the check point status is 'E' the entrant will be marked as excluded by this function. Total time of the entrant will also be updated. For adding normal checkpoints see add_cp.

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.

4.9.2.3 void calc_time (ENTRANT * ent)

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

The time is calculated by subtracting the current time with the start time. Error checking is then done on the result to counter in the cases where the time used is less than a whole hour.

Parameters

<i>ent</i>	Entrant structure used to get the start time, current time and updating the total_time member.
------------	--

4.9.2.4 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. If the entrant has taken a wrong turn somewhere and checked in at a wrong check point -1 is returned.

4.9.2.5 void finished (EVENT * event)

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

Parameters

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

4.9.2.6 **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.

4.9.2.7 **void** `not_started (EVENT * event)`

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

Parameters

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

4.9.2.8 **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.
--------------	------------------

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

Prints out a nicely formatted list of all entrants.

Parameters

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

4.9.2.10 **void** `print_entrant (ENTRANT * entrant)`

Prints out nicely formatted info about a entrant.

Parameters

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

4.9.2.11 **void** `print_entrant_header ()`

Prints out the header used showing what each column is.

ID Name Course Status Start time Total time

4.9.2.12 void print_excluded (EVENT * event, int mc)

Prints out the a list of entrants that has been excluded from the race.

The mc parameter is used to determine if entrants that has been excluded for medical reason should be printed out, or entrants who have been excluded for taking the wrong turn somewhere.

Parameters

<i>event</i>	Event structure containing the list of entrants.
<i>mc</i>	If 1 entrants that has been excluded for medical reasons will be printed out. If 0 entrants which has been excluded for taking the wrong turn will be printed 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](#) ([ENTRANT *ent](#))
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.
- void [add_mc](#) ([EVENT *event](#), char status, [NODE *node](#), [ENTRANT *ent](#), char *time)
This function is used to add a medical check point to a entrant.
- void [print_excluded](#) ([EVENT *event](#), int mc)
Prints out the a list of entrants that has been excluded from the race.

4.10.1 Detailed Description

This file contains definitions of various functions.

Author

Sindre Smistad 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.

The function will create a check point list in the entrant if this is the entrant's first check point. If the check point status is 'I' the entrant will be marked as excluded by this function. Total time of the entrant will also be updated. For adding medical checkpoints see add_mc.

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 add_mc (EVENT * event, char status, NODE * node, ENTRANT * ent, char * time)

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

The function will create a check point list in the entrant if this is the entrant's first check point. If the check point status is 'E' the entrant will be marked as excluded by this function. Total time of the entrant will also be updated. For adding normal checkpoints see add_cp.

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.

4.10.2.3 void calc_time (ENTRANT * ent)

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

The time is calculated by subtracting the current time with the start time. Error checking is then done on the result to counter in the cases where the time used is less than a whole hour.

Parameters

<i>ent</i>	Entrant structure used to get the start time, current time and updating the total_time member.
------------	--

4.10.2.4 `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. If the entrant has taken a wrong turn somewhere and checked in at a wrong check point -1 is returned.

4.10.2.5 `void finished (EVENT * event)`

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

Parameters

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

4.10.2.6 `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.

4.10.2.7 `void not_started (EVENT * event)`

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

Parameters

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

4.10.2.8 `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.
--------------	------------------

4.10.2.9 void print_all (EVENT * event)

Prints out a nicely formatted list of all entrants.

Parameters

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

4.10.2.10 void print_excluded (EVENT * event, int mc)

Prints out the a list of entrants that has been excluded from the race.

The mc parameter is used to determine if entrants that has been excluded for medical reason should be printed out, or entrants who have been excluded for taking the wrong turn somewhere.

Parameters

<i>event</i>	Event structure containing the list of entrants.
<i>mc</i>	If 1 entrants that has been excluded for medical reasons will be printed out. If 0 entrants which has been excluded for taking the wrong turn will be printed out.

4.10.2.11 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.

4.10.2.12 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.

4.10.2.13 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

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 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 i 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 inputed 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`

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 main.c File Reference

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



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

Functions

- int **main** ()

4.13.1 Detailed Description

Author

Sindre Smistad sis13@aber.ac.uk

Date

2012.11.20

4.14 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**

4.14.1 Detailed Description

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

Author

Sindre Smistad sis13@aber.ac.uk

Date

2012.11.25

4.14.2 Function Documentation**4.14.2.1 `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.

4.14.2.2 `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.

4.15 `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.

4.15.1 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

Date

2012.11.25

4.15.2 Typedef Documentation

4.15.2.1 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.

4.15.2.2 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.

4.15.3 Function Documentation

4.15.3.1 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.

4.15.3.2 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.

4.16 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** ()

4.16.1 Detailed Description

This file contains the definitions of several functi.

Author

Sindre Smistad sis13@aber.ac.uk

on 24 November 2012, 02:09

4.16.2 Function Documentation

4.16.2.1 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");
```

```
}
```

4.17 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.

4.17.1 Detailed Description

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

Author

Sindre Smistad sis13@aber.ac.uk

Date

2012.12.07

4.18 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.

Variables

- `int errno`

4.18.1 Detailed Description

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

Author

Sindre Smistad sis13@aber.ac.uk

Date

2012.11.30

4.18.2 Function Documentation

4.18.2.1 `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

<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.

4.18.2.2 `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.

4.19 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.

4.19.1 Detailed Description

This file contains the definition of the track structure.

Author

Sindre Smistad sis13@aber.ac.uk

Date

2012.11.24

4.19.2 Typedef Documentation

4.19.2.1 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.

4.19.2.2 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.

4.19.3 Function Documentation

4.19.3.1 `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, [31](#)
- add_mc
 - functions.c, [28](#)
 - functions.h, [31](#)
- ask_int
 - input.c, [35](#)
 - input.h, [38](#)
- ask_str
 - input.c, [35](#)
 - input.h, [38](#)
- COURSE
 - course.h, [17](#)
- COURSE_LIST
 - course.h, [17](#)
- calc_time
 - functions.c, [28](#)
 - functions.h, [31](#)
- 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, [45](#)
- 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, [8](#)
 - excluded, [8](#)
 - mc_excluded, [8](#)
 - 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](#)
 - print_entrant_header, [24](#)
 - read_entrants, [24](#)
- entrant_finished
 - functions.c, [28](#)
 - functions.h, [31](#)
- entrant_id_search
 - entrant.c, [21](#)
 - entrant.h, [23](#)
- entrant_list, [9](#)
- 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](#)
- excluded
 - entrant, [8](#)
- finished
 - functions.c, [28](#)
 - functions.h, [32](#)
- format_time
 - functions.c, [29](#)
 - functions.h, [32](#)

- functions.c, 26
 - add_cp, 27
 - add_mc, 28
 - calc_time, 28
 - entrant_finished, 28
 - finished, 28
 - format_time, 29
 - not_started, 29
 - out_track, 29
 - print_all, 29
 - print_entrant, 29
 - print_entrant_header, 29
 - print_excluded, 30
- functions.h, 30
 - add_cp, 31
 - add_mc, 31
 - calc_time, 31
 - entrant_finished, 31
 - finished, 32
 - format_time, 32
 - not_started, 32
 - out_track, 32
 - print_all, 32
 - print_excluded, 33
 - read_event_data, 33
 - read_tracks, 33
 - update_status, 33
- get_courses
 - input.c, 35
 - input.h, 38
- get_cp_data
 - input.c, 35
 - input.h, 39
- get_entrants
 - input.c, 35
 - input.h, 39
- get_event
 - input.c, 36
 - input.h, 39
- get_nodes
 - input.c, 36
 - input.h, 39
- get_tracks
 - input.c, 36
 - input.h, 39
- input.c, 34
 - ask_int, 35
 - ask_str, 35
 - get_courses, 35
 - get_cp_data, 35
 - get_entrants, 35
 - get_event, 36
 - get_nodes, 36
 - get_tracks, 36
 - load_cp_data, 36
 - query_entrant, 37
 - show_menu, 37
- input.h, 37
 - ask_int, 38
 - ask_str, 38
 - get_courses, 38
 - get_cp_data, 39
 - get_entrants, 39
 - get_event, 39
 - get_nodes, 39
 - get_tracks, 39
 - load_cp_data, 40
 - query_entrant, 40
 - show_menu, 40
- load_cp_data
 - input.c, 36
 - input.h, 40
- main.c, 40
- mc_excluded
 - entrant, 8
- NODE
 - node.h, 43
- NODE_LIST
 - node.h, 43
- name
 - event, 10
- node, 10
 - node_id, 11
- node.c, 41
 - node_id_search, 42
 - read_nodes, 42
- node.h, 42
 - NODE, 43
 - NODE_LIST, 43
 - node_id_search, 43
 - read_nodes, 44
- node_id
 - node, 11
- node_id_search
 - node.c, 42
 - node.h, 43
- node_list, 11
 - course_list, 6
- not_started
 - functions.c, 29
 - functions.h, 32
- out_track
 - functions.c, 29
 - functions.h, 32
- print_all
 - functions.c, 29
 - functions.h, 32
- print_entrant
 - entrant.h, 24
 - functions.c, 29
- print_entrant_header

- entrant.h, 24
- functions.c, 29
- print_excluded
 - functions.c, 30
 - functions.h, 33
- query_entrant
 - input.c, 37
 - input.h, 40
- 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, 33
- read_nodes
 - node.c, 42
 - node.h, 44
- read_tracks
 - functions.h, 33
 - track.c, 46
- search_course_id
 - course.c, 16
 - course.h, 18
- show_menu
 - input.c, 37
 - input.h, 40
- status
 - entrant, 8
- TRACK
 - track.h, 47
- TRACK_LIST
 - track.h, 47
- tests.h, 44
 - course_read, 45
- time_struct, 11
- time_struct.h, 45
- track, 12
- track.c, 45
 - read_tracks, 46
 - validate_track, 46
- track.h, 47
 - TRACK, 47
 - TRACK_LIST, 47
 - validate_track, 48
- track_list, 12
 - course_list, 6
- update_status
 - entrant.c, 22
 - functions.h, 33
- validate_course
 - course.c, 16
 - course.h, 18
- validate_track
 - track.c, 46
 - track.h, 48