AAU Scheduler

1.0

Generated by Doxygen 1.8.10

Mon Dec 14 2015 17:52:31

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 Course Struct Reference

The **Course** (p. 5) struct contains information about a specific course.

`#include <structs.h>`

Data Fields

- char **name** [64]
- int **totLectures**
- int **numTeachers**
- struct **Teacher** ∗∗ **teachers**

3.1.1 Detailed Description

The **Course** (p. 5) struct contains information about a specific course.

3.1.2 Field Documentation

3.1.2.1 name

The name of the teacher

3.1.2.2 numTeachers

Number of teachers associated

3.1.2.3 teachers

Array of associated teachers

3.1.2.4 totLectures

The total amount of lectures on the course

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

- **structs.h**

3.2 Flags Struct Reference

The **Flags** (p. 6) struct contains a list of flags used to prevent double calculation of fitness.

`#include <structs.h>`

Data Fields

- int **doubleBookingRoom**
- int **doubleBookingLecture**
- int **lectureCounted**
- int **semesterCounted**

3.2.1 Detailed Description

The **Flags** (p. 6) struct contains a list of flags used to prevent double calculation of fitness.

3.2.2 Field Documentation

3.2.2.1 doubleBookingLecture

This flag is set when a lecture has been calculated for overlapping with another lecture from the same specialization

3.2.2.2 doubleBookingRoom

This flag is set when a lecture has been calculated for room doublebooking

3.2.2.3 lectureCounted

This flag is set when the lecture has been counted

3.2.2.4 semesterCounted

This flag is set when the semester that the lecure is part of is counted

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

- **structs.h**

3.3 Generation Struct Reference

The **Generation** (p. 6) struct contains an array of schedules in the generation and a pointer to **SemesterData** (p. 10) which contains relevant information.

`#include <structs.h>`

Data Fields

- int **fitness**
- struct **Schedule schedules** [**GENERATION_SIZE**]
- struct **SemesterData** * **sd**

3.3.1 Detailed Description

The **Generation** (p. 6) struct contains an array of schedules in the generation and a pointer to **SemesterData** (p. 10) which contains relevant information.

3.3.2 Field Documentation

3.3.2.1 fitness

Combined fitness of generation

3.3.2.2 schedules

Array of schedules

3.3.2.3 sd

Pointer to **SemesterData** (p. 10), which contains relevant information

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

- **structs.h**

3.4 Lecture Struct Reference

The **Lecture** (p. 7) struct contains information about a specific lecture.

`#include <structs.h>`

Data Fields

- int **day**
- int **period**
- struct **Room** ∗ **assignedRoom**
- struct **Course** ∗ **assignedCourse**
- struct **Flags flags**
- int **fitness**

3.4.1 Detailed Description

The **Lecture** (p. 7) struct contains information about a specific lecture.

3.4.2 Field Documentation

3.4.2.1 assignedCourse

Pointer to the course the lecture is part of

3.4.2.2 assignedRoom

Pointer to the room assigned to the lecture

3.4.2.3 day

The day the lecture is on

3.4.2.4 fitness

Last calculated fitness value

3.4.2.5 flags

Link to lecture flags

3.4.2.6 period

The period of the day the lecture is on

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

- **structs.h**

3.5 OffTime Struct Reference

The **OffTime** (p. 8) struct contains a day and time period (0 or 1) where the teacher isn't available for a teacher.

`#include <structs.h>`

Data Fields

- int **day**
- int **periods** [**MAX_PERIODS**]

3.5.1 Detailed Description

The **OffTime** (p. 8) struct contains a day and time period (0 or 1) where the teacher isn't available for a teacher.

3.5.2 Field Documentation

3.5.2.1 day

A specific day the **OffTime** (p. 8) effects

3.5.2.2 periods

Array of effected periods

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

- **structs.h**

3.6 Room Struct Reference

The **Room** (p. 9) struct contains the name of a specific room.

`#include <structs.h>`

Data Fields

- char **name** [32]
- int **seats**

3.6.1 Detailed Description

The **Room** (p. 9) struct contains the name of a specific room.

3.6.2 Field Documentation

3.6.2.1 name

The name of the room

3.6.2.2 seats

The number of seats in the room

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

- **structs.h**

3.7 Schedule Struct Reference

The **Schedule** (p. 9) struct contains all lectures for a given time spand.

`#include <structs.h>`

Data Fields

- struct **Generation** ∗ **parentGen**
- struct **Lecture** ∗ **lectures**
- int **fitness**

3.7.1 Detailed Description

The **Schedule** (p. 9) struct contains all lectures for a given time spand.

3.7.2 Field Documentation

3.7.2.1 fitness

Last calculated fitness value

3.7.2.2 lectures

Array of lectures

3.7.2.3 parentGen

Pointer to the generation that this schedule belongs to

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

- **structs.h**

3.8 SemesterData Struct Reference

The **SemesterData** (p. 10) struct contains all available information about a specific semester.

`#include <structs.h>`

Data Fields

- int **numWeeks**
- int **numRooms**
- struct **Room** ∗ **rooms**
- int **numTeachers**
- struct **Teacher** ∗ **teachers**
- int **numCourses**
- struct **Course** ∗ **courses**
- int **numSpecializations**
- struct **Specialization** ∗ **specializations**
- int **numLectures**

3.8.1 Detailed Description

The **SemesterData** (p. 10) struct contains all available information about a specific semester.

A generation would be built with an amount of **SemesterData** (p. 10) structs

3.8.2 Field Documentation

3.8.2.1 courses

Array of assigned courses

3.8.2.2 numCourses

The number of courses assigned to the semester

3.8.2.3 numLectures

The number of lectures in each schedule

3.8.2.4 numRooms

The number of rooms assigned to the semester

3.8.2.5 numSpecializations

The number of specializations assigned to the semester

3.8.2.6 numTeachers

The number of teachers assigned to the semester

3.8.2.7 numWeeks

Total amount of weeks in the semester

3.8.2.8 rooms

Array of assigned rooms

3.8.2.9 specializations

Array of assigned specializations

3.8.2.10 teachers

Array of assiged teachers

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

- **structs.h**

3.9 Specialization Struct Reference

The **Specialization** (p. 11) struct contains information about a specific specialization.

`#include <structs.h>`

Data Fields

- char **name** [32]
- int **numStudents**
- int **numCourses**
- struct **Course** ∗∗ **courses**

3.9.1 Detailed Description

The **Specialization** (p. 11) struct contains information about a specific specialization.

3.9.2 Field Documentation

3.9.2.1 courses

Array of associated courses

3.9.2.2 name

The name of the specialization

3.9.2.3 numCourses

The number of courses on the specialization

3.9.2.4 numStudents

The number of students on the specialization

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

- **structs.h**

3.10 Teacher Struct Reference

The **Teacher** (p. 12) struct contains information about a specific teacher.

`#include <structs.h>`

Data Fields

- char **name** [32]
- int **numOffTimes**
- struct **OffTime** * **offTimes**

3.10.1 Detailed Description

The **Teacher** (p. 12) struct contains information about a specific teacher.

3.10.2 Field Documentation

3.10.2.1 name

The name of the teacher

3.10.2.2 numOffTimes

The number of OffTimes the teacher has_denorm

3.10.2.3 offTimes

Array of OffTimes

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

- **structs.h**

Chapter 4

File Documentation

4.1 data_utility.c File Reference

A set of utility functions when parts of code can be used more than once.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "data_utility.h"
#include "structs.h"
#include "defs.h"
#include "fitness_calculation.h"
```

Functions

- void **initialize_generation** (**Generation** ∗∗gen, **SemesterData** ∗sd)

  Allocate memory for all schedules in a generation.
- void **initialize_schedule** (**Generation** ∗parentGen, int scheduleIndex)

  Allocate memory for lectures in a schedule and sets parent generation.
- void **copy_schedule** (**Schedule** ∗dest, **Schedule** ∗src)

  *Creates a deep copy of a **Schedule** (p. 9).*
- void **copy_generation** (**Generation** ∗dest, **Generation** ∗src)

  *Creates a deep copy of a **Generation** (p. 6).*
- void **print_schedule_issues** (**Schedule** ∗schedule)

  Print the issues with a lecture.
- void **reset_schedule_flags** (**Schedule** ∗schedule)

  Reset flags for all lectures in a specific schedule.
- void **set_lecture** (**Lecture** ∗lect, int day, int period, **Room** ∗room, **Course** ∗course)

  Set members of a lecture struct.
- int **teacher_has_offtime** (**SemesterData** ∗sd, **Teacher** ∗teacher, int dayId, int periodId)

  Check if a teacher has offtime on a given day and period.
- int **specialization_has_lecture** (**Specialization** ∗sp, **Lecture** ∗lect)

  Checks if a specialization has a given lecture.
- int **get_students_on_course** (**SemesterData** ∗sd, **Course** ∗course)

  Gets the total number of students on a given course.
- void **calc_amount_of_lectures** (**SemesterData** ∗sd)

  *Sets the total amount of lectures for a **SemesterData** (p. 10).*
- int **get_specializations_on_course** (**SemesterData** ∗sd, **Course** ∗course, **Specialization** ∗∗∗specs)

Gets all specializations that contains a specific course.

- void **free_generation** (**Generation** ∗gp)

  Brief.

- void **free_semesterdata** (**SemesterData** ∗sd)

  *Free all memory associated with the **SemesterData** (p. 10) struct. Dynamically allocated arrays inside the structs are also freed.*

- const char ∗ **get_name_of_period** (int periodId)

  Gets a name of a period (08:15-12:00 or 12:30-16:15)

- const char ∗ **get_name_of_day** (int dayId)

  Gets a name of a day.

Variables

- const char ∗ **periodNames** [ ] = {"08:15 - 12:00", "12:30 - 16:15"}
- const char ∗ **dayNames** [ ] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}

4.1.1 Detailed Description

A set of utility functions when parts of code can be used more than once.

4.1.2 Function Documentation

4.1.2.1 void calc_amount_of_lectures (SemesterData ∗ *sd*)

Sets the total amount of lectures for a **SemesterData** (p. 10).

Parameters

in	*sd*	**SemesterData** (p. 10) contains the information required to get the amount

Goes through all the courses in **SemesterData** (p. 10) and adds the lectures assigned to each course to a variable which is then returned

4.1.2.2 void copy_generation (Generation ∗ *dest,* Generation ∗ *src*)

Creates a deep copy of a **Generation** (p. 6).

Parameters

in	*dest*	Pointer to the destination **Generation** (p. 6)
in	*src*	Pointer to the source **Generation** (p. 6)

4.1.2.3 void copy_schedule (Schedule ∗ *dest,* Schedule ∗ *src*)

Creates a deep copy of a **Schedule** (p. 9).

Parameters

in	*dest*	Pointer to the destination **Schedule** (p. 9)
in	*src*	Pointer to the source **Schedule** (p. 9)

4.1.2.4 void free_generation (Generation ∗ *gp*)

Brief.

Parameters

in	*gp*	Parameter_Description

Returns

Return_Description

Details

4.1.2.5 void free_semesterdata (SemesterData * *sd*)

Free all memory associated with the **SemesterData** (p. 10) struct. Dynamically allocated arrays inside the structs are also freed.

Parameters

in	*sd*	Parameter_Description

Returns

Return_Description

Details

4.1.2.6 const char* get_name_of_day (int *dayId*)

Gets a name of a day.

Parameters

in	*dayId*	The ID of the day to check

Returns

Returns the name of a day specified by the dayId parameter

4.1.2.7 const char* get_name_of_period (int *periodId*)

Gets a name of a period (08:15-12:00 or 12:30-16:15)

Parameters

in	*periodId*	The ID of the period to check

Returns

Returns the name of a period specified by the periodId parameter

Checks if the period ID is outside the range. If it is outside the range it returns an UNKNOWN name. Otherwise it returns the name according to it's place on the schedule

4.1.2.8 int get_specializations_on_course (SemesterData * *sd,* Course * *course,* Specialization *** *specs*)

Gets all specializations that contains a specific course.

Parameters

in	*sd*	**SemesterData** (p. 10) contains information required for the function to work
in	*course*	The course we check for
out	*specs*	The specializations that contain the course

Returns

Returns amount of specializations

Counts and returns the specializations with the specific cause.

4.1.2.9 int get_students_on_course (SemesterData * *sd*, Course * *course*)

Gets the total number of students on a given course.

Parameters

in	*sd*	**SemesterData** (p. 10) has some required information
in	*course*	Pointer to the couse we check

Returns

Returns the amount of students on a specific course

First we go through all specializations and add them to a temp. variable. Then we go through all the courses in these specializations and adds the number of students in each specialization that has the course

4.1.2.10 void initialize_generation (Generation ** *gen*, SemesterData * *sd*)

Allocate memory for all schedules in a generation.

Parameters

in	*gen*	The generation that contains the schedules
in	*sd*	**SemesterData** (p. 10) contains information needed by the function

Allocates memory for the generation and adds schedules to the gen variable

4.1.2.11 void initialize_schedule (Generation * *parentGen*, int *scheduleIndex*)

Allocate memory for lectures in a schedule and sets parent generation.

Parameters

in	*parentGen*	Pointer to the parent generation
in	*scheduleIndex*	The index of the current schedule

Allocates memory for the lectures and then sets the parent generation in the schedule to the given generation through parentGen

4.1.2.12 void print_schedule_issues (Schedule * *schedule*)

Print the issues with a lecture.

Parameters

in	*schedule*	Pointer to the schedule that has issues

Resets schedule flags, gets fitness values and prints any issues should there be any

4.1.2.13 void reset_schedule_flags (Schedule * *schedule*)

Reset flags for all lectures in a specific schedule.

Parameters

in	*schedule*	**Schedule** (p. 9) contains the lectures we want to reset

Iterates through all lectures in a given schedule and resets all the flags

4.1.2.14 void set_lecture (Lecture * *lect,* int *day,* int *period,* Room * *room,* Course * *course*)

Set members of a lecture struct.

Parameters

in	*lect*	The lecture we want to work with
in	*day*	The new value for day
in	*period*	The new value for period
in	*room*	The new value for assignedRoom
in	*course*	The new value for assignedCourse

4.1.2.15 int specialization_has_lecture (Specialization * *sp,* Lecture * *lect*)

Checks if a specialization has a given lecture.

Parameters

in	*sp*	Pointer to a specialization we want to check
in	*lect*	Pointer to a lecture we want to check with

Returns

Returns whether a specialization is on a specific lecture

Check if a course in the specialization matches the assigned course for the lecture

4.1.2.16 int teacher_has_offtime (SemesterData * *sd,* Teacher * *teacher,* int *dayId,* int *periodId*)

Check if a teacher has offtime on a given day and period.

Parameters

in	*sd*	**SemesterData** (p. 10) contains some information needed by the function
in	*teacher*	Pointer to the teacher we are checking
in	*dayId*	The ID of the day we check for
in	*periodId*	The ID of the period we check for

Returns

Returns 1 if a teacher has an offtime (not available) on a day and period

First we validate the period on the day. Then we iterate through all offtimes

4.1.3 Variable Documentation

4.1.3.1 char * dayNames[ ] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}

The names of the weekdays of a standart work week

4.1.3.2 char * periodNames[ ] = {"08:15 - 12:00", "12:30 - 16:15"}

The names of the two periods on a day

4.2 data_utility.h File Reference

This file contains prototypes required by the **data_utility.c** (p. 15) script.

```
#include "structs.h"
```

Functions

- void **initialize_generation** (**Generation** **gen, **SemesterData** *sd)
  Allocate memory for all schedules in a generation.
- void **initialize_schedule** (**Generation** *parentGen, int scheduleIndex)
  Allocate memory for lectures in a schedule and sets parent generation.
- void **copy_generation** (**Generation** *dest, **Generation** *src)
  *Creates a deep copy of a **Generation** (p. 6).*
- void **copy_schedule** (**Schedule** *dest, **Schedule** *src)
  *Creates a deep copy of a **Schedule** (p. 9).*
- void **print_schedule_issues** (**Schedule** *schedule)
  Print the issues with a lecture.
- void **reset_schedule_flags** (**Schedule** *schedule)
  Reset flags for all lectures in a specific schedule.
- void **set_lecture** (**Lecture** *lect, int day, int period, **Room** *room, **Course** *course)
  Set members of a lecture struct.
- int **teacher_has_offtime** (**SemesterData** *sd, **Teacher** *teacher, int dayId, int periodId)
  Check if a teacher has offtime on a given day and period.
- int **specialization_has_lecture** (**Specialization** *sp, **Lecture** *lect)
  Checks if a specialization has a given lecture.
- int **get_students_on_course** (**SemesterData** *sd, **Course** *course)
  Gets the total number of students on a given course.
- void **calc_amount_of_lectures** (**SemesterData** *sd)
  *Sets the total amount of lectures for a **SemesterData** (p. 10).*
- int **get_specializations_on_course** (**SemesterData** *sd, **Course** *course, **Specialization** ***specs)
  Gets all specializations that contains a specific course.
- void **free_generation** (**Generation** *gp)
  Brief.
- void **free_semesterdata** (**SemesterData** *sd)
  *Free all memory associated with the **SemesterData** (p. 10) struct. Dynamically allocated arrays inside the structs are also freed.*
- const char * **get_name_of_period** (int periodId)
  Gets a name of a period (08:15-12:00 or 12:30-16:15)
- const char * **get_name_of_day** (int dayId)
  Gets a name of a day.

4.2.1 Detailed Description

This file contains prototypes required by the **data_utility.c** (p. 15) script.

4.2.2 Function Documentation

4.2.2.1 void calc_amount_of_lectures (SemesterData * sd)

Sets the total amount of lectures for a **SemesterData** (p. 10).

Parameters

in	*sd*	**SemesterData** (p. 10) contains the information required to get the amount

Goes through all the courses in **SemesterData** (p. 10) and adds the lectures assigned to each course to a variable which is then returned

4.2.2.2 void copy_generation (Generation * dest, Generation * src)

Creates a deep copy of a **Generation** (p. 6).

Parameters

in	*dest*	Pointer to the destination **Generation** (p. 6)
in	*src*	Pointer to the source **Generation** (p. 6)

4.2.2.3 void copy_schedule (Schedule * dest, Schedule * src)

Creates a deep copy of a **Schedule** (p. 9).

Parameters

in	*dest*	Pointer to the destination **Schedule** (p. 9)
in	*src*	Pointer to the source **Schedule** (p. 9)

4.2.2.4 void free_generation (Generation * gp)

Brief.

Parameters

in	*gp*	Parameter_Description

Returns

Return_Description

Details

4.2.2.5 void free_semesterdata (SemesterData * sd)

Free all memory associated with the **SemesterData** (p. 10) struct. Dynamically allocated arrays inside the structs are also freed.

Parameters

in	*sd*	Parameter_Description

Returns

Return_Description

Details

4.2.2.6 const char* get_name_of_day (int *dayId*)

Gets a name of a day.

Parameters

in	*dayId*	The ID of the day to check

Returns

Returns the name of a day specified by the dayId parameter

4.2.2.7 const char* get_name_of_period (int *periodId*)

Gets a name of a period (08:15-12:00 or 12:30-16:15)

Parameters

in	*periodId*	The ID of the period to check

Returns

Returns the name of a period specified by the periodId parameter

Checks if the period ID is outside the range. If it is outside the range it returns an UNKNOWN name. Otherwise it returns the name according to it's place on the schedule

4.2.2.8 int get_specializations_on_course (SemesterData * *sd*, Course * *course*, Specialization *** *specs*)

Gets all specializations that contains a specific course.

Parameters

in	*sd*	**SemesterData** (p. 10) contains information required for the function to work
in	*course*	The course we check for
out	*specs*	The specializations that contain the course

Returns

Returns amount of specializations

Counts and returns the specializations with the specific cause.

4.2.2.9 int get_students_on_course (SemesterData * *sd*, Course * *course*)

Gets the total number of students on a given course.

Parameters

in	*sd*	**SemesterData** (p. 10) has some required information
in	*course*	Pointer to the couse we check

Returns

Returns the amount of students on a specific course

First we go through all specializations and add them to a temp. variable. Then we go through all the courses in these specializations and adds the number of students in each specialization that has the course

4.2.2.10 void initialize_generation (Generation ∗∗ *gen,* SemesterData ∗ *sd*)

Allocate memory for all schedules in a generation.

Parameters

in	*gen*	The generation that contains the schedules
in	*sd*	**SemesterData** (p. 10) contains information needed by the function

Allocates memory for the generation and adds schedules to the gen variable

4.2.2.11 void initialize_schedule (Generation ∗ *parentGen,* int *scheduleIndex*)

Allocate memory for lectures in a schedule and sets parent generation.

Parameters

in	*parentGen*	Pointer to the parent generation
in	*scheduleIndex*	The index of the current schedule

Allocates memory for the lectures and then sets the parent generation in the schedule to the given generation through parentGen

4.2.2.12 void print_schedule_issues (Schedule ∗ *schedule*)

Print the issues with a lecture.

Parameters

in	*schedule*	Pointer to the schedule that has issues

Resets schedule flags, gets fitness values and prints any issues should there be any

4.2.2.13 void reset_schedule_flags (Schedule ∗ *schedule*)

Reset flags for all lectures in a specific schedule.

Parameters

in	*schedule*	**Schedule** (p. 9) contains the lectures we want to reset

Iterates through all lectures in a given schedule and resets all the flags

4.2.2.14 void set_lecture (Lecture ∗ *lect,* int *day,* int *period,* Room ∗ *room,* Course ∗ *course*)

Set members of a lecture struct.

Parameters

in	*lect*	The lecture we want to work with
in	*day*	The new value for day
in	*period*	The new value for period
in	*room*	The new value for assignedRoom
in	*course*	The new value for assignedCourse

4.2.2.15 int specialization_has_lecture (Specialization * *sp,* Lecture * *lect*)

Checks if a specialization has a given lecture.

Parameters

in	*sp*	Pointer to a specialization we want to check
in	*lect*	Pointer to a lecture we want to check with

Returns

Returns whether a specialization is on a specific lecture

Check if a course in the specialization matches the assigned course for the lecture

4.2.2.16 int teacher_has_offtime (SemesterData * *sd,* Teacher * *teacher,* int *dayId,* int *periodId*)

Check if a teacher has offtime on a given day and period.

Parameters

in	*sd*	**SemesterData** (p. 10) contains some information needed by the function
in	*teacher*	Pointer to the teacher we are checking
in	*dayId*	The ID of the day we check for
in	*periodId*	The ID of the period we check for

Returns

Returns 1 if a teacher has an offtime (not available) on a day and period

First we validate the period on the day. Then we iterate through all offtimes

4.3 defs.h File Reference

This file contains the defines required by the program.

Macros

- #define **BUFFER_SIZE** 512
- #define **WEEK_WIDTH** 10.0f
- #define **TABLE_WIDTH** 100.0f
- #define **MAX_PERIODS** 2
- #define **DAYS_PER_WEEK** 5
- #define **GENERATION_SIZE** 500
- #define **MAX_GENERATIONS** 200
- #define **MUTATION_CHANCE** 10

- #define **ERROR_OUT_OF_MEMORY** 1
- #define **ERROR_ARRAY_BOUNDS_EXCEEDED** 2
- #define **ERROR_FILE_NULL_PTR** 3
- #define **MAX_OVER_CAPACITY** 1.05
- #define **MAX_LECTURES_PER_WEEK** 7
- #define **PENALTY_ROOM_TOO_BIG** 25
- #define **PENALTY_ROOM_TOO_SMALL** 50
- #define **PENALTY_DOUBLEBOOKING** 200
- #define **PENALTY_TEACHER_BOOKED** 200
- #define **PENALTY_TEACHER_OFFTIME** 200
- #define **PENALTY_DAILY_LIMIT** 25
- #define **PENALTY_WEEKLY_LIMIT** 10
- #define **PENALTY_SEMESTER_DISTRIB** 50
- #define **MIN**(a, b) (((a)<(b))?(a):(b))
- #define **MAX**(a, b) (((a)>(b))?(a):(b))

4.3.1 Detailed Description

This file contains the defines required by the program.

4.3.2 Macro Definition Documentation

4.3.2.1 #define BUFFER_SIZE 512

The size of the string buffer to hold the content from the read file

4.3.2.2 #define DAYS_PER_WEEK 5

The amount of days on one week

4.3.2.3 #define ERROR_ARRAY_BOUNDS_EXCEEDED 2

Sent in the event that we exceed the bounds of an array

4.3.2.4 #define ERROR_FILE_NULL_PTR 3

Sent in the event that we cannot find a requested file

4.3.2.5 #define ERROR_OUT_OF_MEMORY 1

Sent in the event that we cannot allocate required memory

4.3.2.6 #define GENERATION_SIZE 500

The amount of Schedules in one **Generation** (p. 6)

4.3.2.7 #define MAX(*a, b*) (((a)>(b))?(a):(b))

Computes the maximum of a and b

4.3.2.8 #define MAX_GENERATIONS 200

The maximum amount of generations till the program stops trying

4.3.2.9 #define MAX_LECTURES_PER_WEEK 7

The maximum amount of lectures per week

4.3.2.10 #define MAX_OVER_CAPACITY 1.05

The amount of students a room is allowed to be over capacity. For example 5%

4.3.2.11 #define MAX_PERIODS 2

The maximum amount of periods on one day

4.3.2.12 #define MIN(*a, b*) (((a)<(b))?(a):(b))

Computes the minumum of a and b

4.3.2.13 #define MUTATION_CHANCE 10

The chance of a mutation to happen

4.3.2.14 #define PENALTY_DAILY_LIMIT 25

Given if a specific lecture appeares more than once per day

4.3.2.15 #define PENALTY_DOUBLEBOOKING 200

Given in case of room being doublebooked

4.3.2.16 #define PENALTY_ROOM_TOO_BIG 25

Given if the room is larger than the amount of students

4.3.2.17 #define PENALTY_ROOM_TOO_SMALL 50

Given if the room is smaller than the amount of students

4.3.2.18 #define PENALTY_SEMESTER_DISTRIB 50

Given if there are many lectures by the end of a semester

4.3.2.19 #define PENALTY_TEACHER_BOOKED 200

Given in case of a teacher being doublebooked

4.3.2.20 #define PENALTY_TEACHER_OFFTIME 200

Given in case of a teacher being off work

4.3.2.21 #define PENALTY_WEEKLY_LIMIT 10

Given if there are more lectures in one week than a defined number

4.3.2.22 #define TABLE_WIDTH 100.0f

The The width of the table on the web page

4.3.2.23 #define WEEK_WIDTH 10.0f

The space allocated for the width of the week on the web page

4.4 fitness_calculation.c File Reference

This script contains the functions responsible for calculating fitness values for the population.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "structs.h"
#include "data_utility.h"
#include "defs.h"
#include "fitness_calculation.h"
```

Functions

- int **calcfit_capacity** (**SemesterData** ∗sd, **Lecture** ∗lect)

  Calculates how well the lecture fits into the assigned room.
- int **calcfit_teacher_availability** (**Schedule** ∗schedule, **Lecture** ∗lect)

  Calculates fitness based on the teacher's availability at the time of the lecture.
- int **calcfit_doublebooking** (**Schedule** ∗schedule, **Lecture** ∗lect)

  Calculates fitness based on whether the room or period is doublebooked.
- int **calcfit_distribution_weekly** (**Schedule** ∗schedule, **Lecture** ∗lect)

  Calculates how well the lecture fits into its week.
- int **calcfit_distribution_semester** (**Schedule** ∗schedule, **Lecture** ∗lect)

  Calculates the semester distribution.
- int **calcfit_distribution_semester_inner** (**Schedule** ∗schedule, **Lecture** ∗lect, **Specialization** ∗sp)

  Calculate how well the lecture fits into the semester distribution.
- int **calcfit_lecture** (**Schedule** ∗schedule, **Lecture** ∗lect)

  Calculate fitness for a single lecture (gene)
- int **calcfit_schedule** (**Schedule** ∗schedule)

  Calculate fitness for a schedule/genome.
- int **calcfit_generation** (**Generation** ∗gp)

  Calculates the fitness of a generation.

4.4.1 Detailed Description

This script contains the functions responsible for calculating fitness values for the population.

4.4.2 Function Documentation

4.4.2.1 int calcfit_capacity (SemesterData * *sd*, Lecture * *lect*)

Calculates how well the lecture fits into the assigned room.

Parameters

in	*sd*	**SemesterData** (p. 10) contains all the information about the structs needed for this function
in	*lect*	Pointer to lecture to calculate for

Returns

Returns the fitness of the calculation

This function checks the capacity of the room and the amount of students on the lecture and determins the penalty in fitness by comparing the two

4.4.2.2 int calcfit_distribution_semester (Schedule * *schedule*, Lecture * *lect*)

Calculates the semester distribution.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the calculation

Goes through each specialization on the course and calculates their fitness

4.4.2.3 int calcfit_distribution_semester_inner (Schedule * *schedule*, Lecture * *lect*, Specialization * *sp*)

Calculate how well the lecture fits into the semester distribution.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate
in	*sp*	Pointer to a specialization that is needed to obtain some information within the function

Returns

Returns the fitness of the calculation

Goes through all lectures in a week within the specialization and see how well they are distributed. When a lecture is checked, it is flagged as such and will not be checked again

4.4.2.4 int calcfit_distribution_weekly (Schedule * *schedule*, Lecture * *lect*)

Calculates how well the lecture fits into its week.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the lecture distribution

Goes through each lecture and comparing it to another lecture in a schedule. When a lecture has been compared, it is flagged as such and will not be compared again

4.4.2.5 int calcfit_doublebooking (Schedule * *schedule,* Lecture * *lect*)

Calculates fitness based on whether the room or period is doublebooked.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the calculation

Performs calculations for both room and lecture doublebooking

4.4.2.6 int calcfit_generation (Generation * *gp*)

Calculates the fitness of a generation.

Parameters

in	*gp*	The generation to calulate fitness for

4.4.2.7 int calcfit_lecture (Schedule * *schedule,* Lecture * *lect*)

Calculate fitness for a single lecture (gene)

Parameters

in	*schedule*	The schedule the lecture is a part of
in	*lect*	The specific lecture to calculate fitness for

Returns

Returns the fitness of the lecture

Performs all the fitness calculations on a specific lecture and returns the total fitness for that lecture

4.4.2.8 int calcfit_schedule (Schedule * *schedule*)

Calculate fitness for a schedule/genome.

Parameters

in	*schedule*	The schedule to calculate fitness for

Returns

Returns the fitness of the schedule

Iterates through all lectures and add their fitness to a variables which is then returned

4.4.2.9 int calcfit_teacher_availability (**Schedule** * *schedule,* **Lecture** * *lect*)

Calculates fitness based on the teacher's availability at the time of the lecture.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the calculation

Also test whether the teacher is already assigned to a lecture on the same date

4.5 fitness_calculation.h File Reference

This file contains prototypes required by the **fitness_calculation.c** (p. 27) script.

Functions

- int **calcfit_capacity** (**SemesterData** *sd, **Lecture** *lect)
  Calculates how well the lecture fits into the assigned room.
- int **calcfit_teacher_availability** (**Schedule** *schedule, **Lecture** *lect)
  Calculates fitness based on the teacher's availability at the time of the lecture.
- int **calcfit_doublebooking** (**Schedule** *schedule, **Lecture** *lect)
  Calculates fitness based on whether the room or period is doublebooked.
- int **calcfit_distribution_weekly** (**Schedule** *schedule, **Lecture** *lect)
  Calculates how well the lecture fits into its week.
- int **calcfit_distribution_semester** (**Schedule** *schedule, **Lecture** *lect)
  Calculates the semester distribution.
- int **calcfit_distribution_semester_inner** (**Schedule** *schedule, **Lecture** *lect, **Specialization** *sp)
  Calculate how well the lecture fits into the semester distribution.
- int **calcfit_lecture** (**Schedule** *schedule, **Lecture** *lect)
  Calculate fitness for a single lecture (gene)
- int **calcfit_schedule** (**Schedule** *schedule)
  Calculate fitness for a schedule/genome.
- int **calcfit_generation** (**Generation** *gp)
  Calculates the fitness of a generation.

4.5.1 Detailed Description

This file contains prototypes required by the **fitness_calculation.c** (p. 27) script.

4.5.2 Function Documentation

4.5.2.1 int calcfit_capacity (SemesterData * sd, Lecture * lect)

Calculates how well the lecture fits into the assigned room.

Parameters

in	*sd*	**SemesterData** (p. 10) contains all the information about the structs needed for this function
in	*lect*	Pointer to lecture to calculate for

Returns

Returns the fitness of the calculation

This function checks the capacity of the room and the amount of students on the lecture and determins the penalty in fitness by comparing the two

4.5.2.2 int calcfit_distribution_semester (Schedule * schedule, Lecture * lect)

Calculates the semester distribution.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the calculation

Goes through each specialization on the course and calculates their fitness

4.5.2.3 int calcfit_distribution_semester_inner (Schedule * schedule, Lecture * lect, Specialization * sp)

Calculate how well the lecture fits into the semester distribution.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate
in	*sp*	Pointer to a specialization that is needed to obtain some information within the function

Returns

Returns the fitness of the calculation

Goes through all lectures in a week within the specialization and see how well they are distributed. When a lecture is checked, it is flagged as such and will not be checked again

4.5.2.4 int calcfit_distribution_weekly (Schedule * schedule, Lecture * lect)

Calculates how well the lecture fits into its week.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the lecture distribution

Goes through each lecture and comparing it to another lecture in a schedule. When a lecture has been compared, it is flagged as such and will not be compared again

4.5.2.5 int calcfit_doublebooking (Schedule * *schedule,* Lecture * *lect*)

Calculates fitness based on whether the room or period is doublebooked.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the calculation

Performs calculations for both room and lecture doublebooking

4.5.2.6 int calcfit_generation (Generation * *gp*)

Calculates the fitness of a generation.

Parameters

in	*gp*	The generation to calulate fitness for

4.5.2.7 int calcfit_lecture (Schedule * *schedule,* Lecture * *lect*)

Calculate fitness for a single lecture (gene)

Parameters

in	*schedule*	The schedule the lecture is a part of
in	*lect*	The specific lecture to calculate fitness for

Returns

Returns the fitness of the lecture

Performs all the fitness calculations on a specific lecture and returns the total fitness for that lecture

4.5.2.8 int calcfit_schedule (Schedule * *schedule*)

Calculate fitness for a schedule/genome.

Parameters

in	*schedule*	The schedule to calculate fitness for

Returns

Returns the fitness of the schedule

Iterates through all lectures and add their fitness to a variables which is then returned

4.5.2.9 int calcfit_teacher_availability (**Schedule** ∗ *schedule,* **Lecture** ∗ *lect*)

Calculates fitness based on the teacher's availability at the time of the lecture.

Parameters

in	*schedule*	Pointer to a schedule
in	*lect*	Pointer to lecture to calculate

Returns

Returns the fitness of the calculation

Also test whether the teacher is already assigned to a lecture on the same date

4.6 genetic_algorithm.c File Reference

This script contains the functions related to our algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#include "structs.h"
#include "genetic_algorithm.h"
#include "defs.h"
#include "data_utility.h"
#include "fitness_calculation.h"
```

Functions

- void **run_ga** (**Generation** ∗∗curGen, **SemesterData** ∗sd)

  The main function of the algorithm.
- int **ga_select** (**Generation** ∗curGen, **Generation** ∗newGen)

  Select Schedules using Tournament selection.
- void **ga_crossbreed** (**Generation** ∗gp, int carryOver)

  Breed population from Schedules up to carryOver.
- void **ga_mutate** (**Generation** ∗gp)

  *Randomly mutate Schedules in a **Generation** (p. 6).*
- int **compare_schedule_fitness** (const void ∗a, const void ∗b)

  Compares the fitness of two schedules. Used by qsort.

4.6.1 Detailed Description

This script contains the functions related to our algorithm.

4.6.2 Function Documentation

4.6.2.1 int compare_schedule_fitness (const void * *a*, const void * *b*)

Compares the fitness of two schedules. Used by qsort.

Parameters

in	*a*	The first schedule
in	*b*	The second schedule

Returns

Returns a number that tells qsort how to sort the schedules

4.6.2.2 void ga_crossbreed (**Generation** * *gp*, int *carryOver*)

Breed population from Schedules up to carryOver.

Parameters

in	*gp*	Pointer to the generation to crossbreed
in	*carryOver*	The amount of genomes copied from the generation

Finds two parents and compare the fitness of their lectures

4.6.2.3 void ga_mutate (**Generation** * *gp*)

Randomly mutate Schedules in a **Generation** (p. 6).

Parameters

in	*gp*	Pointer to the generation to mutate

Iterates through all lectures in all schedules and mutates randomly

4.6.2.4 int ga_select (**Generation** * *curGen*, **Generation** * *newGen*)

Select Schedules using Tournament selection.

Parameters

in	*curGen*	Pointer to the current generation
in	*newGen*	Pointer to the new generation

Returns

Returns amount of genomes carried over

Schedules (genomes) should be sorted by fitness at this point

4.6.2.5 void run_ga (**Generation** ** *curGen*, **SemesterData** * *sd*)

The main function of the algorithm.

Parameters

in	*curGen*	The current generation
in	*sd*	**SemesterData** (p. 10) contains information needed by the function

Here the algorithm is initiated

4.7 genetic_algorithm.h File Reference

This file contains prototypes required by the **genetic_algorithm.c** (p. 33) script.

Functions

- void **run_ga** (**Generation** ∗∗curGen, **SemesterData** ∗sd)

  The main function of the algorithm.
- int **ga_select** (**Generation** ∗curGen, **Generation** ∗newGen)

  Select Schedules using Tournament selection.
- void **ga_crossbreed** (**Generation** ∗gp, int carryOver)

  Breed population from Schedules up to carryOver.
- void **ga_mutate** (**Generation** ∗gp)

  *Randomly mutate Schedules in a **Generation** (p. 6).*
- int **compare_schedule_fitness** (const void ∗a, const void ∗b)

  Compares the fitness of two schedules. Used by qsort.

4.7.1 Detailed Description

This file contains prototypes required by the **genetic_algorithm.c** (p. 33) script.

4.7.2 Function Documentation

4.7.2.1 int compare_schedule_fitness (const void ∗ *a*, const void ∗ *b*)

Compares the fitness of two schedules. Used by qsort.

Parameters

in	*a*	The first schedule
in	*b*	The second schedule

Returns

Returns a number that tells qsort how to sort the schedules

4.7.2.2 void ga_crossbreed (Generation ∗ *gp*, int *carryOver*)

Breed population from Schedules up to carryOver.

Parameters

in	*gp*	Pointer to the generation to crossbreed
in	*carryOver*	The amount of genomes copied from the generation

Finds two parents and compare the fitness of their lectures

4.7.2.3 void ga_mutate (Generation * *gp*)

Randomly mutate Schedules in a **Generation** (p. 6).

Parameters

in	*gp*	Pointer to the generation to mutate

Iterates through all lectures in all schedules and mutates randomly

4.7.2.4 int ga_select (Generation * *curGen*, Generation * *newGen*)

Select Schedules using Tournament selection.

Parameters

in	*curGen*	Pointer to the current generation
in	*newGen*	Pointer to the new generation

Returns

Returns amount of genomes carried over

Schedules (genomes) should be sorted by fitness at this point

4.7.2.5 void run_ga (Generation ** *curGen*, SemesterData * *sd*)

The main function of the algorithm.

Parameters

in	*curGen*	The current generation
in	*sd*	**SemesterData** (p. 10) contains information needed by the function

Here the algorithm is initiated

4.8 html_output.c File Reference

The html output script is responsible for the html schedules that is being generated.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdarg.h>
#include "defs.h"
#include "structs.h"
#include "data_utility.h"
```

Functions

- void **print_file_header** (FILE *f, char *pageTitle)

  Prints the file header.

- void **print_footer** (FILE ∗f)

  Prints the file footer.
- void **begin_print_table** (FILE ∗f, int cellspacing)

  Initiates a table.
- void **end_print_table** (FILE ∗f)

  Ends a table.
- void **print_row_header** (FILE ∗f, double width, const char ∗str,...)

  Prints a header for a row.
- void **print_title** (FILE ∗f, const char ∗title)

  Prints a shedule title.
- void **begin_print_data** (FILE ∗f, const char ∗str)

  Begins data print.
- void **end_print_data** (FILE ∗f)

  Ends the data print.
- void **begin_print_row** (FILE ∗f, const char ∗backgroundColor)

  Prints the rows of lectures.
- void **end_print_row** (FILE ∗f)

  Ends the row print.
- void **print_period** (**Schedule** ∗schedule, **Specialization** ∗sp, FILE ∗f, int periodId, int weekNumber)

  Prints a period to the schedule.
- void **print_schedule_to_file** (**Schedule** ∗schedule, **Specialization** ∗sp, char ∗fileName)

  Prints a schedule for a specific specialization to a file.

4.8.1 Detailed Description

The html output script is responsible for the html schedules that is being generated.

4.8.2 Function Documentation

4.8.2.1 void begin_print_data (FILE ∗ *f,* const char ∗ *str*)

Begins data print.

Parameters

in	*f*	The file in which the schedule is being generated
in	*str*	The data to be printed

This function is printing the provided data from str into the file f

4.8.2.2 void begin_print_row (FILE ∗ *f,* const char ∗ *backgroundColor*)

Prints the rows of lectures.

Parameters

in	*f*	The file in which the schedule is being generated
in	*backgroundColor*	The color of the row

This function initiates rows with a given color

4.8.2.3 void begin_print_table (FILE ∗ *f,* int *cellspacing*)

Initiates a table.

Parameters

in	*f*	The file in which the schedule is being generated
in	*cellspacing*	The spacing between the cells in the table

This function is laying the foundation for a html table

4.8.2.4 void end_print_data (FILE * *f*)

Ends the data print.

Parameters

in	*f*	The file in which the schedule is being generated

Adds the ending tag for the data

4.8.2.5 void end_print_row (FILE * *f*)

Ends the row print.

Parameters

in	*f*	The file in which the schedule is being generated

This function adds the ending tag for the row

4.8.2.6 void end_print_table (FILE * *f*)

Ends a table.

Parameters

in	*f*	The file in which the schedule is being generated

This function is adding the end table tag for a html table

4.8.2.7 void print_file_header (FILE * *f*, char * *pageTitle*)

Prints the file header.

Parameters

in	*f*	The file in which the schedule is being generated
in	*pageTitle*	The name of the page

This function is responsible for the header of the file

4.8.2.8 void print_footer (FILE * *f*)

Prints the file footer.

Parameters

in	*f*	The file in which the schedule is being generated

This function is responsible for the footer of the file

4.8.2.9 void print_period (Schedule * *schedule*, Specialization * *sp*, FILE * *f*, int *periodId*, int *weekNumber*)

Prints a period to the schedule.

Parameters

in	*schedule*	Pointer to a schedule
in	*sp*	**Specialization** (p. 11) contains information about the specialization the schedule is generated for
in	*f*	The file in which the schedule is being generated
in	*periodId*	Period to print
in	*weekNumber*	The number of the current week

This function adds a period to the schedule and formats it as needed

4.8.2.10 void print_row_header (FILE * *f*, double *width*, const char * *str*, ...)

Prints a header for a row.

Parameters

in	*f*	The file in which the schedule is being generated
in	*width*	The width of the row
in	*str*	The name of the row

This function creates a row with the given width and name

4.8.2.11 void print_schedule_to_file (Schedule * *schedule*, Specialization * *sp*, char * *fileName*)

Prints a schedule for a specific specialization to a file.

Parameters

in	*schedule*	Pointer to a schedule
in	*sp*	**Specialization** (p. 11) contains information about the specialization the schedule is generated for
in	*fileName*	The name of the file in which the schedule should be generated

The final step of the schedule creation

4.8.2.12 void print_title (FILE * *f*, const char * *title*)

Prints a shedule title.

Parameters

in	*f*	The file in which the schedule is being generated
in	*title*	The title

This defines a title for the schedule. An example could be "Schedule for Robotics"

4.9 html_output.h File Reference

This file contains prototypes required by the **html_output.c** (p. 36) script.

Functions

- void **print_schedule_to_file** (**Schedule** *schedule, **Specialization** *sp, char *fileName)
  Prints a schedule for a specific specialization to a file.

4.9.1 Detailed Description

This file contains prototypes required by the **html_output.c** (p. 36) script.

4.9.2 Function Documentation

4.9.2.1 void print_schedule_to_file (Schedule * *schedule,* Specialization * *sp,* char * *fileName*)

Prints a schedule for a specific specialization to a file.

Parameters

in	*schedule*	Pointer to a schedule
in	*sp*	**Specialization** (p. 11) contains information about the specialization the schedule is generated for
in	*fileName*	The name of the file in which the schedule should be generated

The final step of the schedule creation

4.10 input_reader.c File Reference

This script is responsible for reading the data file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "structs.h"
#include "input_reader.h"
#include "defs.h"
```

Functions

- int **read_config** (char *fileName, **SemesterData** *sd)

  Initial function for the config reader.
- void **handle_line** (char *line, **SemesterData** *sd)

  This function handles the lines from the main config reader function.
- int **read_int** (char *line, int *position, int *out)

  Reads an int from a string and adds the amount of digits to position.
- int **read_multiple_words** (char *line, int *position, char *out)

  Reads an entire string between two apostrophes.
- void **add_teacher** (**SemesterData** *sd, char *name, int numOffTimes, **OffTime** *offTimes)

  Adds a teacher to the teachers array.
- void **add_room** (**SemesterData** *sd, char *name, int seats)

  Adds a room to the rooms array.
- void **add_course** (**SemesterData** *sd, char *name, int totLectures, int numTeachers, **Teacher** **teachers)

  Adds a course to the courses array.
- void **add_specialization** (**SemesterData** *sd, char *name, int numStudents, int numCourses, **Course** **courses)

  Adds a specialization to the specializations array.

4.10.1 Detailed Description

This script is responsible for reading the data file.

4.10.2 Function Documentation

4.10.2.1 void add_course (SemesterData * *sd*, char * *name*, int *totLectures*, int *numTeachers*, Teacher ** *teachers*)

Adds a course to the courses array.

Parameters

in	*sd*	The courses array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the course
in	*totLectures*	The total amount of lectures in the course
in	*numTeachers*	The amount of teachers assigned to the course
in	*teachers*	The array of teachers assigned

Allocates the memory needed and updates relevant variables and values

4.10.2.2 void add_room (SemesterData * *sd*, char * *name*, int *seats*)

Adds a room to the rooms array.

Parameters

in	*sd*	The rooms array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the room
in	*seats*	The amount of seats available in the room

Allocates the memory needed and updates relevant variables and values

4.10.2.3 void add_specialization (SemesterData * *sd*, char * *name*, int *numStudents*, int *numCourses*, Course ** *courses*)

Adds a specialization to the specializations array.

Parameters

in	*sd*	The specialization array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the specialization
in	*numStudents*	The total amount of students in the specialization
in	*numCourses*	The amount of courses assigned to the specialization
in	*courses*	The array of courses assigned

Allocates the memory needed and updates relevant variables and values

4.10.2.4 void add_teacher (SemesterData * *sd*, char * *name*, int *numOffTimes*, OffTime * *offTimes*)

Adds a teacher to the teachers array.

Parameters

in	*sd*	The teachers array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the teacher
in	*numOffTimes*	The amount of off times
in	*offTimes*	An array of **OffTime** (p. 8)

Allocates the memory needed and updates relevant variables and values

4.10.2.5 void handle_line (char * *line*, SemesterData * *sd*)

This function handles the lines from the main config reader function.

Parameters

in	*line*	This line is given by the input_reader function
in	*sd*	**SemesterData** (p. 10) is a link to the structs the function needs

This function goes through the line and checks it for commands and parameters. Essentially it works like a parser

4.10.2.6 int read_config (char * *fileName,* SemesterData * *sd*)

Initial function for the config reader.

Parameters

in	*fileName*	The name of the file to read from
in	*sd*	**SemesterData** (p. 10) is a link to our structs that are needed for this function

Returns

Returns 1 or 0 depending whether the function succeded or failed

The function reads the file line by line and formats them to the format we need for further processing, then sends it to handle_line

4.10.2.7 int read_int (char * *line,* int * *position,* int * *out*)

Reads an int from a string and adds the amount of digits to position.

Parameters

in	*line*	The string to read
in	*position*	Current position in the string
out	*out*	A pointer to an int where the final number will be stored

Returns

Returns whether the function has failed or succeded

The function goes through the string (line) until there are no more characters. It then converts the content of the string to int and outputs it to the out variable

4.10.2.8 int read_multiple_words (char * *line,* int * *position,* char * *out*)

Reads an entire string between two apostrophes.

Parameters

in	*line*	The string to read from
in	*position*	The current position in the string
out	*out*	The output string

Returns

Returns whether the function succeded or not

This function reads from the line string and outputs everything between two apostrophes to the output string

4.11 input_reader.h File Reference

This file contains prototypes required by the **input_reader.c** (p. 40) script.

Functions

- int **read_config** (char *fileName, **SemesterData** *data)

  Initial function for the config reader.
- void **handle_line** (char *line, **SemesterData** *data)

  This function handles the lines from the main config reader function.

- int **read_int** (char ∗line, int ∗position, int ∗out)

  Reads an int from a string and adds the amount of digits to position.
- int **read_multiple_words** (char ∗line, int ∗position, char ∗out)

  Reads an entire string between two apostrophes.
- void **add_teacher** (**SemesterData** ∗sd, char ∗name, int numOffTimes, **OffTime** ∗offTimes)

  Adds a teacher to the teachers array.
- void **add_room** (**SemesterData** ∗sd, char ∗name, int seats)

  Adds a room to the rooms array.
- void **add_course** (**SemesterData** ∗sd, char ∗name, int totLectures, int numTeachers, **Teacher** ∗∗teachers)

  Adds a course to the courses array.
- void **add_specialization** (**SemesterData** ∗sd, char ∗name, int numStudents, int numCourses, **Course** ∗∗courses)

  Adds a specialization to the specializations array.

4.11.1 Detailed Description

This file contains prototypes required by the **input_reader.c** (p. 40) script.

4.11.2 Function Documentation

4.11.2.1 void add_course (SemesterData ∗ *sd,* char ∗ *name,* int *totLectures,* int *numTeachers,* Teacher ∗∗ *teachers*)

Adds a course to the courses array.

Parameters

in	*sd*	The courses array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the course
in	*totLectures*	The total amount of lectures in the course
in	*numTeachers*	The amount of teachers assigned to the course
in	*teachers*	The array of teachers assigned

Allocates the memory needed and updates relevant variables and values

4.11.2.2 void add_room (SemesterData ∗ *sd,* char ∗ *name,* int *seats*)

Adds a room to the rooms array.

Parameters

in	*sd*	The rooms array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the room
in	*seats*	The amount of seats available in the room

Allocates the memory needed and updates relevant variables and values

4.11.2.3 void add_specialization (SemesterData ∗ *sd,* char ∗ *name,* int *numStudents,* int *numCourses,* Course ∗∗ *courses*)

Adds a specialization to the specializations array.

Parameters

in	*sd*	The specialization array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the specialization
in	*numStudents*	The total amount of students in the specialization
in	*numCourses*	The amount of courses assigned to the specialization
in	*courses*	The array of courses assigned

Allocates the memory needed and updates relevant variables and values

4.11.2.4 void add_teacher (SemesterData * *sd*, char * *name*, int *numOffTimes*, OffTime * *offTimes*)

Adds a teacher to the teachers array.

Parameters

in	*sd*	The teachers array is part of the **SemesterData** (p. 10) struct
in	*name*	The name of the teacher
in	*numOffTimes*	The amount of off times
in	*offTimes*	An array of **OffTime** (p. 8)

Allocates the memory needed and updates relevant variables and values

4.11.2.5 void handle_line (char * *line*, SemesterData * *sd*)

This function handles the lines from the main config reader function.

Parameters

in	*line*	This line is given by the input_reader function
in	*sd*	**SemesterData** (p. 10) is a link to the structs the function needs

This function goes through the line and checks it for commands and parameters. Essentially it works like a parser

4.11.2.6 int read_config (char * *fileName*, SemesterData * *sd*)

Initial function for the config reader.

Parameters

in	*fileName*	The name of the file to read from
in	*sd*	**SemesterData** (p. 10) is a link to our structs that are needed for this function

Returns

Returns 1 or 0 depending whether the function succeded or failed

The function reads the file line by line and formats them to the format we need for further processing, then sends it to handle_line

4.11.2.7 int read_int (char * *line*, int * *position*, int * *out*)

Reads an int from a string and adds the amount of digits to position.

Parameters

in	*line*	The string to read
in	*position*	Current position in the string

out	*out*	A pointer to an int where the final number will be stored

Returns

Returns whether the function has failed or succeded

The function goes through the string (line) until there are no more characters. It then converts the content of the string to int and outputs it to the out variable

4.11.2.8 int read_multiple_words (char * *line*, int * *position*, char * *out*)

Reads an entire string between two apostrophes.

Parameters

in	*line*	The string to read from
in	*position*	The current position in the string
out	*out*	The output string

Returns

Returns whether the function succeded or not

This function reads from the line string and outputs everything between two apostrophes to the output string

4.12 scheduler.c File Reference

The main script of the program, the magic starts here.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <assert.h>
#include "structs.h"
#include "scheduler.h"
#include "input_reader.h"
#include "data_utility.h"
#include "fitness_calculation.h"
#include "defs.h"
#include "html_output.h"
#include "genetic_algorithm.h"
```

Functions

- int **main** (void)

  The starting point of the program.

4.12.1 Detailed Description

The main script of the program, the magic starts here.

4.12.2 Function Documentation

4.12.2.1 int main (void)

The starting point of the program.

Returns

Returns whether the program has exited with an error or success

In here, the function that reads the config file is run and then the genetic algorithm is run. Lastly, the schedules are being generated in html documents and ready to be implemented to a web interface for example

4.13 scheduler.h File Reference

This file contains prototypes required by the **scheduler.c** (p. 47) script.

Functions

- void **free_generation** (**Generation** ∗gp)

  Brief.
- void **free_semesterdata** (**SemesterData** ∗sd)

  *Free all memory associated with the **SemesterData** (p. 10) struct. Dynamically allocated arrays inside the structs are also freed.*
- int **compare_schedule_fitness** (const void ∗a, const void ∗b)

  Compares the fitness of two schedules. Used by qsort.
- void **initialize_schedule** (**Generation** ∗parentGen, int scheduleIndex)

  Allocate memory for lectures in a schedule and sets parent generation.

4.13.1 Detailed Description

This file contains prototypes required by the **scheduler.c** (p. 47) script.

4.13.2 Function Documentation

4.13.2.1 int compare_schedule_fitness (const void ∗ *a*, const void ∗ *b*)

Compares the fitness of two schedules. Used by qsort.

Parameters

in	*a*	The first schedule
in	*b*	The second schedule

Returns

Returns a number that tells qsort how to sort the schedules

4.13.2.2 void free_generation (Generation ∗ *gp*)

Brief.

Parameters

in	*gp*	Parameter_Description

Returns

Return_Description

Details

4.13.2.3 void free_semesterdata (SemesterData * *sd*)

Free all memory associated with the **SemesterData** (p. 10) struct. Dynamically allocated arrays inside the structs are also freed.

Parameters

in	*sd*	Parameter_Description

Returns

Return_Description

Details

4.13.2.4 void initialize_schedule (Generation * *parentGen,* int *scheduleIndex*)

Allocate memory for lectures in a schedule and sets parent generation.

Parameters

in	*parentGen*	Pointer to the parent generation
in	*scheduleIndex*	The index of the current schedule

Allocates memory for the lectures and then sets the parent generation in the schedule to the given generation through parentGen

4.14 structs.h File Reference

The header file containing all the structs required by the program.

```
#include "defs.h"
```

Data Structures

- struct **Room**

  *The **Room** (p. 9) struct contains the name of a specific room.*
- struct **Teacher**

  *The **Teacher** (p. 12) struct contains information about a specific teacher.*
- struct **OffTime**

  *The **OffTime** (p. 8) struct contains a day and time period (0 or 1) where the teacher isn't available for a teacher.*
- struct **Course**

  *The **Course** (p. 5) struct contains information about a specific course.*
- struct **Specialization**

  *The **Specialization** (p. 11) struct contains information about a specific specialization.*

- struct **Flags**

  *The **Flags** (p. 6) struct contains a list of flags used to prevent double calculation of fitness.*
- struct **Lecture**

  *The **Lecture** (p. 7) struct contains information about a specific lecture.*
- struct **SemesterData**

  *The **SemesterData** (p. 10) struct contains all available information about a specific semester.*
- struct **Schedule**

  *The **Schedule** (p. 9) struct contains all lectures for a given time spand.*
- struct **Generation**

  *The **Generation** (p. 6) struct contains an array of schedules in the generation and a pointer to **SemesterData** (p. 10) which contains relevant information.*

Typedefs

- typedef struct **Room Room**
- typedef struct **OffTime OffTime**
- typedef struct **Teacher Teacher**
- typedef struct **Course Course**
- typedef struct **Specialization Specialization**
- typedef struct **Flags Flags**
- typedef struct **Lecture Lecture**
- typedef struct **SemesterData SemesterData**
- typedef struct **Schedule Schedule**
- typedef struct **Generation Generation**

4.14.1 Detailed Description

The header file containing all the structs required by the program.

Index