P1: Automatiser dit studievalg

Generated by Doxygen 1.8.13

Contents

Chapter 1

Coding Style

General

- Indentation: 4 times whitespace per indent
- Language: English (both code and comments)
- Brackets: None if only one argument

### Functions

```
void func(param1, param2){
    function();
}
```

Structures

```
int i;
for(i = 0; i < 9; i++)
    function();
```

```
int i;
for(i = 0; i < 9; i++){
    function();
    function1();
}
```

```
if(this && that)
    function();
else if(just this)
    function7();
else
    function1();
```

```
if(this && that){
    function();
    function1();
} else if(just this){
    function7();
    function();
} else{
    function1();
    function2();
}
```

```
while(this is true)
    function();
```

```
while(this is true){
    function();
    function1();
}
```

```
do
    function();
while(that);
```

```
do{
    function();
    function1();
} while(that);
```

```
switch(expression){
    case 1:
        function1();
        break;
    case 2:
        function2();
        break;
    default:
        return shit;
}
```

### Operators

```
i = 3 + 4 * (7 / 5 + 5 % 2);
i += 2;
i -= 2;
```

### Structures

```
char array[SYMBOLIC_CONSTANT] = "Some text";
char *array = "Some other text";
char array[] = "Some text again";
```

variablenames are written with underscores:

```
int this_integer = 2;
char some_character = 'a';
```

Calls

Functionnames are written in camelcase:

```
callFunction(a, 2, 3, a, bdw, w);
parseSomething(a, b);
```

Chapter 2

Main Program

Basic functions

Input af datasæt

Funktionen skal finde en liste af alle forskellige interesser, der kommer fra datasæt (separat fil). Funktionen skal finde en liste over alle uddannelser, fra filen Funktionen skal gemme vægtningerne for alle uddannelser og interesser Gemme via outputparametre

Brugerinput

Læser data fra terminalen, som brugeren indtaster Holder dialogen med brugeren i gang Får interesser fra "Input af datasæt", som den stiller spørgsmål om

Output brugervektor

Funktionen gemmer brugerens interrese-vektor til en fil seperat fra data-filen

Vektorregning

Vektorer skal gemmes som arrays af doubles funktioner returnere gemmen outputvektorer Skal kunne udføre følgende udregninger:

- Add
  - Addition af to vektorer
  - Skal også kunne bruges til substraktion
- Scale
  - Skal kunne skalere en vektor
- Length
  - Skal kunne beregne længden af en vektor ved brug af Pythagoras
- Normalize
  - Skal kunne omdanne en vektor til den tilsvarende enhedsvektor (længde = 1)
- Dot Product
  - Skal kunne bestemme prikprodukt hvis vektorne er enhedsvektorer
  - Udregnes som: x_1x_2 + y_1y_2 + z_1z_2

Beregning af output

Funktionen skal kunne sammenligne brugerens vektor med uddannelsernes vektorerne og bestemme den, der passer bedst Bruger prikprodukt og en adjustment vektor (som er den, der justerer brugerens vektor efter tidligere evalueringer)

Formatering af output

Funktionen skal skrive navnet samt information fra datasættet.

Input evaluering

Funktion skal beregne adjustment vektoren ud fra brugerens evaluering af den viste uddannelse.

$0.1 < k < 0.5$ adjust_vector = adjust_vector + normalize(study_vector - user_vector) $*$ k $*$ eval

Command handling

Funktionen skal finde ud af hvilken command brugeren har indtastet samt om der hører argumenter med, hvilket også gemmes. Følgende commands skal håndteres:

- Find uddannelse
- Save uddannelse
- Recommend uddannelse
- Load saved
- Evaluate uddannelse

Interactions

Funktionen skal fungere som kerne-funktionen der kalder på de andre funktioner og holder på de variabler som programmet gemmer undervejs.

Chapter 3

Navngivning

En uddannelse kalder vi `bachelor` En vektor af interesser kalder vi `interst[]`

```
struct bachelor{
  double interest[];
}
```

Filstrukturen af databasen

```
Interesse:            .... \t  .... \t  .... \t  .... \t
Fag:                  mat  \t  dan  \t  .... \t
           Biologi    0.3  \t       \t       \t
           Datalogi   1.0  \t  0.0  \t       \t        \t beskrivelse \t  region  \t  'z'  \t  'A' (krav)
           Dansk      0.0  \t  1.0
           ...
           ..
           ...
```

Til at beskrive fagenes niveau bruge ASCII code points for bogstaverne 'A', 'B' og 'C'. Bogstavet 'Z' .. .

Chapter 4

P1

Automatiser dit studievalg A302

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

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

Chapter 7

Class Documentation

7.1 CuString Struct Reference

Public Attributes

- int **length**
- int **size**
- char ∗ **buffer**

7.1.1 Detailed Description

Definition at line 20 of file CuTest.h.

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

- include/CuTest.h

7.2 CuSuite Struct Reference

Public Attributes

- int **count**
- CuTest ∗ **list** [MAX_TEST_CASES]
- int **failCount**

7.2.1 Detailed Description

Definition at line 98 of file CuTest.h.

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

- include/CuTest.h

7.3 CuTest Struct Reference

Public Attributes

- char ∗ **name**
- TestFunction **function**
- int **failed**
- int **ran**
- const char ∗ **message**
- jmp_buf ∗ **jumpBuf**

7.3.1 Detailed Description

Definition at line 43 of file CuTest.h.

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

- include/CuTest.h

7.4 database Struct Reference

Public Attributes

- int **amount_of_educations**
- struct education ∗ educations

  the amount of educations in the database
- int amount_of_interests

  an array of educations delimited by amount_of_educations
- char ∗∗ interest_string

  the amount of interests in the database

7.4.1 Detailed Description

Definition at line 16 of file database.h.

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

- include/database.h

7.5 Database Struct Reference

A structure to store a database.

```
#include <database.h>
```

7.5.1 Detailed Description

A structure to store a database.

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

- include/database.h

7.6 education Struct Reference

Describes an education and all it requirements.

```
#include <education.h>
```

Public Attributes

- char ∗ **name**
- char ∗ **description**
- char ∗ **link**
- enum region **region**
- double **required_grade**
- struct vector **interests**
- struct qualification **required_qualifications**

7.6.1 Detailed Description

Describes an education and all it requirements.

A structure, which contains amount_of_educations educations.

This structure defines an education and all the details about the education.

Definition at line 27 of file education.h.

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

- include/education.h

7.7 location Struct Reference

Public Attributes

- enum region **region**
- double **region_importance**

7.7.1 Detailed Description

Definition at line 24 of file region.h.

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

- include/region.h

7.8 profile Struct Reference

Describes a user.

```
#include <profile.h>
```

Public Attributes

- struct vector **interests**
- struct vector **adjustment_vector**
- char **name** [MAX_NAME_LENGTH]
- struct qualification **qualifications**
- double **average**
- struct location **location**
- char **saved_educations** [EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH]
- int **last_recommended**
- char **recommended_educations** [EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH]

7.8.1 Detailed Description

Describes a user.

This structure defines the profile of a user and all the details about the user

Definition at line 31 of file profile.h.

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

- include/profile.h

7.9 qualification Struct Reference

Public Attributes

- int **amount_of_subjects**
- struct subject ∗ subjects

  the amount of subjects in qualifications

7.9.1 Detailed Description

Definition at line 75 of file subjects.h.

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

- include/subjects.h

7.10 subject Struct Reference

Public Attributes

- enum level level

  the name of the subject

7.10.1 Detailed Description

Definition at line 66 of file subjects.h.

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

- include/subjects.h

7.11 vector Struct Reference

Public Attributes

- double ∗ **array**
- int **size**

7.11.1 Detailed Description

Definition at line 14 of file vector.h.

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

- include/vector.h

Chapter 8

File Documentation

8.1 include/commands.h File Reference

Contains functions related to command handling.

```
#include "profile.h"
#include "education.h"
#include "subjects.h"
#include "vector.h"
#include "database.h"
```

Functions

- void menuCmd (void)

    Prints all the possible commands the user can use.
- void surveyCmd (struct profile ∗user, const struct database ∗db)

    Tests the current user for name, location, interests, qualifications and average grade.
- void setProfileLocation (struct profile ∗user)

    Sets the region of choice in user.
- double convertScale (int initial_value)

    Returns the converted value.
- int validScaleValue (int value, int interval_start, int interval_end)

    Returns a value between interval_start and interval_end.
- int getValidInteger (void)

    Returns a valid integer given through the terminal.
- void setProfileInterests (struct profile ∗user, const struct database ∗db)

    Saves all interests to user as a converted value (see convertScale)
- void setProfileQualifications (struct profile ∗user)

    Saves all the users qualifications as given by the terminal.
- void setSubjects (struct profile ∗user)

    Sets all qualifications in user to match the enum class.
- void setImportantSubjects (struct profile ∗user)

    Saves all the qualifications for the important subjects.
- const char ∗ classNameStr (enum class class)

    Returns the name as a string of a class given as an enum class.

- enum level levelAsValue (char c)

    Returns the enum value of a level given as a character.
- void setOtherSubjects (struct profile ∗user, int start, int end)

    Saves all the levels of the other subjects (not the important ones)
- void chooseFromList (struct profile ∗user, int interval_start, int interval_end)

    Saves the levels of chosen subjects to user.
- double getValidDouble (void)

    Returns a valid double entered in the terminal.
- void evalCmd (struct profile ∗user, struct education ∗current_education, int arg)

    Changes the adjustment vector for the user to approach the current education.
- struct education findCmd (char ∗arg, const struct database ∗db)

    Finds and prints out the education with the exact name given as and argument.
- void searchCmd (char ∗arg, const struct database ∗db)

    Finds and prints out the educations whose name contains the given argument.
- struct education recommendCmd (struct profile ∗user, const struct database ∗database)

    Goes trough the available educations and compares them to the user: Both their interests, qualifications and location are considered.
- int isQualified (struct profile user, struct education education)

    Checks if the user has the subject levels required by the education.
- const char ∗ getRegionName (enum region r)

    Returns the name of the region as a string.
- void **printEducation** (struct education)
- void saveCmd (struct profile ∗user, struct education ∗current_education)

    Saves the given education to a list in the profile struct.
- int getIndex (char edu_array[EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH], struct education target)

    Returns the index of the given target in the array.
- int getEmptyIndex (char edu_array[EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH])

    Returns an index with an empty string in the given array.
- int listIsFull (int i)

    A logical statement that returns a boolean value.
- void clearBuffer (void)

    Empties the buffer for standard input.
- void listCmd (const struct profile ∗user)

    Prints out the names of all the saved educations.
- void deleteCmd (struct profile ∗user, int deleted_entry)

    Removes the name of the education at the given index.
- void saveProfile (struct profile user)

    Saves a file with the information collected about the user.
- int **checkProfile** (const char name[ ])
- struct profile loadProfile (char ∗name, int number_of_interests)

    Loads a user profile from a generated <name>_profile.txt file.

8.1.1 Detailed Description

Contains functions related to command handling.

Contains all of the functions used for handling commands, such as those relating to verifying input and the functions that act on receiving a command.

8.1.2 Function Documentation

8.1.2.1 chooseFromList()

```
void chooseFromList (
            struct profile * user,
            int interval_start,
            int interval_end )
```

Saves the levels of chosen subjects to user.

Parameters

user	The profile struct where the qualifications should be saved
interval_start	The start of the interval for the qualifications in the list
interval_end	The end of the interval for the qualifications in the list

Definition at line 257 of file commands.c.

```
                                                                                      {
    int temp_subject, i = 0, scan_res, length_string;
    char temp_char;
    char temp_string[MAX_INPUT_LENGTH];

    fgets(temp_string, MAX_INPUT_LENGTH - 1, stdin);
    length_string = strlen(temp_string);

    if(length_string < 2)
        return;


    do{
        scan_res = sscanf(temp_string + i, " %d%c", &temp_subject, &temp_char);
        if(temp_subject >= 0 && temp_subject < (interval_end - interval_start + 1) &&
      levelAsValue(temp_char) != -1  && scan_res == 2){
            user->qualifications.subjects[temp_subject + interval_start].
      level = levelAsValue(temp_char);
            i += 1;
            while(isalnum(*(temp_string + ++i)) == 0);
        }
    } while(i < length_string && scan_res != 0);
}
```

8.1.2.2 classNameStr()

```
const char * classNameStr (
            enum class class  )
```

Returns the name as a string of a class given as an enum class.

Parameters

class	The enum value the name should return for

Definition at line 201 of file commands.c.

```
201                                           {
202    char *classes[TOTAL_SUBJECTS + USELESS_SUBJECTS] = {"MATHEMATICS", "CHEMISTRY", "BIOLOGY", "PHYSICS", "
      ENGLISH",
203                                      "BIOTECHNOLOGY", "GEOSCIENCE", "HISTORY", "IDEA_HISTORY",
204                                      "INFORMATICS", "INTERNATIONAL_ECONOMICS", "COMMUNICATION_AND_IT",
205                                      "RELIGION", "SOCIALSTUDIES", "BUSINESS_ECONOMICS", "
      CONTEMPORAY_HISTORY",
206                                      "FRENCH", "SPANISH", "GERMAN", "CHINESE", "ARABIC", "GREEK", "ITALIAN"
      ,
207                                      "JAPANESE", "LATIN", "PORTUGESE", "RUSSIAN", "NONE", "DANISH"};
208    return classes[class];
209 }
```

8.1.2.3 convertScale()

```
double convertScale (
              int v )
```

Returns the converted value.

Parameters

v	The value to be converted

Returns

A double value between -1 and 1 given that the input is between 0 and 10

Definition at line 99 of file commands.c.

```
99                          {
100    return (((double) v - 5.0) / 5.0);
101 }
```

8.1.2.4 deleteCmd()

```
void deleteCmd (
              struct profile * user,
              int deleted_entry )
```

Removes the name of the education at the given index.

Parameters

user	The profile struct for the user

Definition at line 550 of file commands.c.

```
                                                        {
    strcpy(user->saved_educations[validScaleValue(deleted_entry, 0, EDUCATION_LIST_LENGTH)],
   "");
}
```

8.1.2.5 evalCmd()

```
void evalCmd (
               struct profile * user,
               struct education * current_education,
               int arg )
```

Changes the adjustment vector for the user to approach the current education.

The distance of the change is determined by the argument

Parameters

current_education	The education currently being displayed
user	The profile struct whose adjustment vector is changed
arg	The user input argument for how much to change the adjustment vector

Definition at line 348 of file commands.c.

```
                                                                                {
    struct vector user_vector = addVector(user->interests, user->adjustment_vector);
    struct vector distance_vector = subtractVector(current_education->interests,
  user_vector);
    struct vector scale_vector = scaleVector(distance_vector, ADJUSTMENT_CONSTANT *
  convertScale(arg));

    user->adjustment_vector = addVector(user->adjustment_vector, scale_vector);

    freeVectorM(3, user_vector, distance_vector, scale_vector);
}
```

8.1.2.6 findCmd()

```
struct education findCmd (
               char * arg,
               const struct database * db )
```

Finds and prints out the education with the exact name given as and argument.

Parameters

arg	The argument string which should be the name of an education
database	The database in which all educations are stored

Returns

A struct for the education found

Definition at line 303 of file commands.c.

```
                                                            {
    int i, edu_found = 0;
    struct education edu;
    for(i = 0; i < db->amount_of_educations; i++){
        if(strcmp(arg, db->educations[i].name) == 0){
            edu = db->educations[i];
            edu_found = 1;
        }
    }
    if(edu_found)
        printEducation(edu);
    else
        printf("No education exists by that name\n");
    return edu;
}
```

8.1.2.7 getEmptyIndex()

```
int getEmptyIndex (
              char edu_array[EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH] )
```

Returns an index with an empty string in the given array.

Parameters

edu_array	An array of strings in which the empty string should be found

Definition at line 495 of file commands.c.

```
                                                                             {
    int i = 0;
    int index = NO_EMPTY_INDEX;

    for(i = 0; index == NO_EMPTY_INDEX && i < EDUCATION_LIST_LENGTH; i++){
        if(strcmp(edu_array[i], "") == 0){
            index = i;
        }
    }

    return index;
}
```

8.1.2.8 getIndex()

```
int getIndex (
              char edu_array[EDUCATION_LIST_LENGTH][MAX_EDU_NAME_LENGTH],
              struct education target )
```

Returns the index of the given target in the array.

Parameters

edu_array	An array of strings
target	An education whose name is to be found in the array

Definition at line 478 of file commands.c.

```
                                                                                                          {
    int i = 0;
    int index = NOT_IN_LIST;

    for(i = 0; index == NOT_IN_LIST && i < EDUCATION_LIST_LENGTH; i++){
        if(strcmp(edu_array[i], target.name) == 0){
            index = i;
        }
    }

    return index;
}
```

8.1.2.9 getRegionName()

```
const char * getRegionName (
              enum region r )
```

Returns the name of the region as a string.

Parameters

r	The enum region value of the region to be returned as a string

Definition at line 438 of file commands.c.

```
                                        {
    switch(r){
        case NORTH_JUTLAND:
            return "North Jutland";
        case CENTRAL_JUTLAND:
            return "Central Jutland";
        case SOUTHERN_DENMARK:
            return "Southern Denmark";
        case ZEALAND:
            return "Zealand";
        case CAPITAL_REGION:
            return "Capital Region";
        default:
            return "Region Not Found";
    }
}
```

8.1.2.10 isQualified()

```
int isQualified (
              struct profile user,
              struct education education )
```

Checks if the user has the subject levels required by the education.

Parameters

user	The profile struct whose quailification is checked
education	The education struct with the requirements

Returns

0 if the user does not have the required levels and 1 if the user does

Definition at line 400 of file commands.c.

```
                                                                {
    int i;
    struct subject subject;
    for(i = 0; i < education.required_qualifications.amount_of_subjects; i++){
        subject = education.required_qualifications.subjects[i];
        if(user.qualifications.subjects[subject.name].level <
  subject.level)
            return 0;
    }
    return 1;
}
```

8.1.2.11 levelAsValue()

```
enum level levelAsValue (
              char c )
```

Returns the enum value of a level given as a character.

Parameters

c	The level as a character to be converted to enum level

Definition at line 215 of file commands.c.

```
                               {
    enum level return_value = -1;

    switch(c){
        case 'A': case 'a':
            return_value = A;
            break;
        case 'B': case 'b':
            return_value = B;
            break;
        case 'C': case 'c':
            return_value = C;
            break;
        case 'Z': case 'z':
            return_value = Z;
            break;
        default:
            return_value = -1;
    }
    return return_value;
}
```

8.1.2.12 listCmd()

```
void listCmd (
            const struct profile * user )
```

Prints out the names of all the saved educations.

Parameters

user	The profile struct for the user

Definition at line 531 of file commands.c.

```
                                        {
    int i, counter = 0;

    printf("\nList of saved educations:\n");
    for(i = 0; i < EDUCATION_LIST_LENGTH; i++){
        if(strcmp(user->saved_educations[i], "") != 0){
            printf("%2d: %s\n", i, user->saved_educations[i]);
            counter++;
        }
    }

    if(counter == 0)
        printf("No entries yet\n\n");
}
```

8.1.2.13 listIsFull()

```
int listIsFull (
            int i )
```

A logical statement that returns a boolean value.

Parameters

i	The index of an array of education structs 1 if the index is -1 and 0 otherwise

Definition at line 513 of file commands.c.

```
                       {
    return i == NO_EMPTY_INDEX;
}
```

8.1.2.14 loadProfile()

```
struct profile loadProfile (
            char * name,
            int number_of_interests )
```

Loads a user profile from a generated <name>_profile.txt file.

Parameters

char	∗name The name of the user
int	number_of_interests The number of interests which is a member of the database struct

Returns

struct profile Returns a user profile

Definition at line 615 of file commands.c.

```
                                                              {
    int i;
    FILE *file_pointer;
    struct profile user;
    char file_name[MAX_FILE_NAME_LENGTH];
    char version[MAX_FILE_NAME_LENGTH];
    char buffer[MAX_INPUT_LENGTH] = "Ingenting";

    sprintf(file_name, "%s_profil.txt", name);

    file_pointer = fopen(file_name, "r");

    if(file_pointer == NULL){
        printf("File could not be opened");
        exit(EXIT_FAILURE);
    }

    user = createProfile(number_of_interests);

    fscanf(file_pointer, "%s %s %f %d %f\n", version, user.name, &user.average, &user.location.region, &
  user.location.region_importance);

    fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
    for (i = 0; i < EDUCATION_LIST_LENGTH; i++){
        fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
        sscanf(buffer, "%[^\n]s", user.saved_educations[i]);
    }

    fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
    for (i = 0; i < EDUCATION_LIST_LENGTH; i++){
        fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
        sscanf(buffer, "%[^\n]s", user.recommended_educations[i]);
    }

    fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
    for (i = 0; i < user.interests.size; i++){
        fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
        sscanf(buffer, "%lf", &user.interests.array[i]);
    }

    fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
    for (i = 0; i < user.adjustment_vector.size; i++){
        fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
        sscanf(buffer, "%lf", &user.adjustment_vector.array[i]);
    }

    fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
    for (i = 0; i < user.qualifications.amount_of_subjects; i++){
        fgets(buffer, MAX_INPUT_LENGTH, file_pointer);
        sscanf(buffer, "%d", &user.qualifications.subjects[i].level);
    }

    fclose(file_pointer);

    printf("Profile successfully loaded.\n\n");

    return user;
}
```

8.1.2.15 recommendCmd()

```
struct education recommendCmd (
            struct profile * user,
            const struct database * database )
```

Goes trough the available educations and compares them to the user: Both their interests, qualifications and location are considered.

Parameters

user	The profile struct which is compared
database	The database containing the educations

Returns

A struct for the recommended education.

Definition at line 365 of file commands.c.

```
                                                                                    {
    int i;
    struct vector normalized_vector;
    double highest_result = -3.0, result = 0.0;
    struct education best_fit;
    normalized_vector = normalizeVector(addVector(user->interests, user->
  adjustment_vector));

    for(i = 0; i < database->amount_of_educations; i++){
        result = dotProduct(database->educations[i].interests, normalized_vector) +
                 (1.0 - (double) abs(user->location.region - database->
  educations[i].region)) *
                  user->location.region_importance;
        if(result > highest_result && isQualified(*user, database->
  educations[i]) &&
           getIndex(user->recommended_educations, database->educations[i]) == NOT_IN_LIST
  ){
            highest_result = result;
            best_fit = database->educations[i];
        }
    }

    freeVector(normalized_vector);

    strcpy(user->recommended_educations[user->last_recommended], best_fit.name);
    user->last_recommended = (user->last_recommended + 1) % EDUCATION_LIST_LENGTH;

    printf("\nThe recommended education is:");
    printEducation(best_fit);

    return best_fit;
}
```

8.1.2.16 saveCmd()

```
void saveCmd (
            struct profile * user,
            struct education * current_education )
```

Saves the given education to a list in the profile struct.

Parameters

current_education	A pointer to an education
user	The profile struct of the user in which the education is saved

Definition at line 460 of file commands.c.

```
460                                                                           {
461     int i;
462 
463     i = getEmptyIndex(user->saved_educations);
464 
465     if(getIndex(user->saved_educations, *current_education) != -1)
466         printf("Already in list\n");
467     else if(listIsFull(i))
468         printf("List empty. Use delete to delete entries\n");
469     else
470         strcpy(user->saved_educations[i], current_education->name);
471 }
```

8.1.2.17 saveProfile()

```
void saveProfile (
              struct profile user )
```

Saves a file with the information collected about the user.

Parameters

user	The profile struct for the user

Definition at line 558 of file commands.c.

```
558                                     {
559     FILE *file_pointer;
560     int i;
561     char file_name[MAX_FILE_NAME_LENGTH];
562     sprintf(file_name, "%s_profil.txt", user.name);
563 
564     file_pointer = fopen(file_name, "w");
565 
566     if(file_pointer != NULL){                       /* Checks if file could be opened */
567         fprintf(file_pointer, "%s %s %f %d %f\n", VERSION, user.name, user.average, user.location.region,
      user.location.region_importance);
568 
569         fprintf(file_pointer, "Saved\n");
570         for (i = 0; i < EDUCATION_LIST_LENGTH; i++)
571             fprintf(file_pointer, "%s\n", user.saved_educations[i]);
572 
573         fprintf(file_pointer, "Recommend\n");
574         for (i = 0; i < EDUCATION_LIST_LENGTH; i++)
575             fprintf(file_pointer, "%s\n", user.recommended_educations[i]);
576 
577         fprintf(file_pointer, "Interests\n");
578         for (i = 0; i < user.interests.size; i++)
579             fprintf(file_pointer, "%f\n", user.interests.array[i]);
580 
581         fprintf(file_pointer, "Adjustment\n");
582         for (i = 0; i < user.adjustment_vector.size; i++)
583             fprintf(file_pointer, "%f\n", user.adjustment_vector.array[i]);
584 
585         fprintf(file_pointer, "Qualifications\n");
586         for(i = 0; i < TOTAL_SUBJECTS; i++)
```

```
            fprintf(file_pointer, "%d\n", user.qualifications.subjects[i].
    level);
    } else{
        printf("File could not be opened");
        exit(EXIT_FAILURE);
    }
    fclose(file_pointer);

    printf("File saved successfully\n\n");
}
```

8.1.2.18 searchCmd()

```
void searchCmd (
            char * arg,
            const struct database * db )
```

Finds and prints out the educations whose name contains the given argument.

Parameters

arg	The argument string which should be contained in the name of an education
database	The database in which all educations are stored.

Definition at line 325 of file commands.c.

```
                                                       {
    int i, edu_found = 0;
    struct education edu;

    for(i = 0; i < db->amount_of_educations; i++){
        if(strstr(db->educations[i].name, arg) != NULL) {
            printf("    %s\n", db->educations[i].name);
            edu_found = 1;
        }
    }
    if(edu_found)
        printf("Use the \"find\" command to look up your desired education\n");
    else
        printf("No education exists by that name\n");
}
```

8.1.2.19 setImportantSubjects()

```
void setImportantSubjects (
            struct profile * user )
```

Saves all the qualifications for the important subjects.

Parameters

user	The profile struct where the subjects are saved to

Definition at line 183 of file commands.c.

```
                                                {
    char temp_char;
    int i;

    for(i = 0; i < IMPORTANT_SUBJECTS; i++){
        printf("%s:%*s ", classNameStr(i), (int) (FIELD_SIZE - strlen(
  classNameStr(i))), "");
        do{
            scanf(" %c", &temp_char);
        } while(levelAsValue(temp_char) == -1);
        user->qualifications.subjects[i].level = levelAsValue(temp_char);
        clearBuffer();
    }
}
```

8.1.2.20 setOtherSubjects()

```
void setOtherSubjects (
              struct profile * user,
              int start,
              int end )
```

Saves all the levels of the other subjects (not the important ones)

Parameters

user	The profile struct where the qualifications are to be saved
start	The start of the subjects to be asked for
end	The ens of the subjects to be asked for

Definition at line 243 of file commands.c.

```
                                                                {
    int i;

    for(i = 0; i < end - start; i++)
        printf("%d: %s\n", i, classNameStr(i + start));
    chooseFromList(user, start, end);
}
```

8.1.2.21 setProfileInterests()

```
void setProfileInterests (
              struct profile * user,
              const struct database * db )
```

Saves all interests to user as a converted value (see convertScale)

Parameters

user	The profile struct where the interests are saved to
db	The database struct where information about all interests are saved as a pointer

Definition at line 136 of file commands.c.

```
                                                                        {
  int i;

  printf("Next, a series of interests will be shown\n"
          "You are to give a value between 0 and 10, "
          "where 0 is negative and 10 is positive towards the interest\n");

  for(i = 0; i < db->amount_of_interests; i++){
      printf("%s:%*s ", db->interest_string[i], (int) (FIELD_SIZE - strlen(db->
   interest_string[i])), "");
      user->interests.array[i] = convertScale(validScaleValue(
   getValidInteger(), 0, 10));
  }
  printf("\n\n\n");
}
```

8.1.2.22 setProfileLocation()

```
void setProfileLocation (
              struct profile * user )
```

Sets the region of choice in user.

Saves the interest in studying in this location

Parameters

user	The profile struct where the information about location should be saved

Definition at line 81 of file commands.c.

```
                                              {
  int i;

  printf("Where do you want to study?\n");
  for(i = 0; i < NUMBER_OF_REGIONS; i++)
      printf("%d: %s    ", i, getRegionName(i));
  printf("\n");
  user->location.region = validScaleValue(getValidInteger(), 0,
   NUMBER_OF_REGIONS - 1);

  printf("How important is this region to you\n");
  user->location.region_importance = (convertScale(validScaleValue(
   getValidInteger(), 0, 10)) + 1.0) / 2.0;
}
```

8.1.2.23 setProfileQualifications()

```
void setProfileQualifications (
              struct profile * user )
```

Saves all the users qualifications as given by the terminal.

Parameters

user	The profile struct where the qualifications are saved to

Definition at line 154 of file commands.c.

```
                                                    {
    setSubjects(user);

    printf("Your qualifications regarding subjects from high school will now be tested\n"
           "Give a level from A, B, C or Z if you have not had the subject\n");

    setImportantSubjects(user);

    printf("Now some less relevant subjects will be \n");

    setOtherSubjects(user, IMPORTANT_SUBJECTS, IMPORTANT_SUBJECTS + OTHER_SUBJECTS);
    setOtherSubjects(user, IMPORTANT_SUBJECTS + OTHER_SUBJECTS, TOTAL_SUBJECTS);
}
```

8.1.2.24 setSubjects()

```
void setSubjects (
              struct profile * user )
```

Sets all qualifications in user to match the enum class.

Parameters

user	The profile struct where the subjects are saved to

Definition at line 172 of file commands.c.

```
                                         {
    int i;

    for(i = 0; i < TOTAL_SUBJECTS; i++)
        user->qualifications.subjects[i].name = i;
}
```

8.1.2.25 surveyCmd()

```
void surveyCmd (
              struct profile * user,
              const struct database * db )
```

Tests the current user for name, location, interests, qualifications and average grade.

Parameters

user	The profile struct where all survey results are saved
db	The database where information of interests and subjects are as a pointer

Definition at line 40 of file commands.c.

```
40                                                              {
41     char name[MAX_NAME_LENGTH], choice;
42 
43     /*  Introduction  */
44     printf("This survey will ask you several questions about interests, qualifications and grades\n"
45            "The survey requires answers in numbers (integers), and where scale is part, a value between 1
       and 100\n\n");
46 
47     /*  Scan for profile name  */
48     printf("Profile name: ");
49     scanf("%s", name);
50     if(checkProfile(name) == 1){
51         printf("Profile name is in use. Stop survey? (Y/N)\n");
52         scanf(" %c", &choice);
53         if(choice == 'Y' || choice == 'y')
54             return;
55     }
56     strcpy(user->name, name);
57 
58     /*  Get location and assesment  */
59     setProfileLocation(user);
60 
61     /*  Get all interests  */
62     setProfileInterests(user, db);
63 
64     /*  Get all qualifications  */
65     setProfileQualifications(user);
66 
67     /*  Get average grade  */
68     printf("What is your average grade? ");
69     user->average = getValidDouble();
70 
71     /*  Ending the survey  */
72     printf("The survey has now concluded. Returning to menu...\n\n");
73 }
```

8.1.2.26 validScaleValue()

```
int validScaleValue (
              int value,
              int interval_start,
              int interval_end )
```

Returns a value between interval_start and interval_end.

If the given value outside the interval it will return the value inside the interval closest the value. The interval_start must be less than the interval_end

Parameters

value	The value to check within the scale
interval_start	The start value of the scale
interval_end	The end value the scale

Definition at line 111 of file commands.c.

```
111                                                                      {
112     return (value > interval_end ? interval_end : (value < interval_start ? interval_start : value));
113 }
```

8.2 include/constants.h File Reference

Contains symbolic constants used throughout the program.

Macros

- #define **VERSION** "1.0.1"
- #define **NUMBER_OF_REGIONS** 5
- #define **IMPORTANT_SUBJECTS** 5
- #define **OTHER_SUBJECTS** 11
- #define **LANGUAGE_SUBJECTS** 11
- #define **USELESS_SUBJECTS** 2
- #define **TOTAL_SUBJECTS** (IMPORTANT_SUBJECTS + OTHER_SUBJECTS + LANGUAGE_SUBJEC↩ TS)
- #define **MAX_NAME_LENGTH** 20
- #define **MAX_FILE_NAME_LENGTH** MAX_NAME_LENGTH + 12
- #define **EDUCATION_LIST_LENGTH** 10
- #define **MAX_EDU_NAME_LENGTH** 40
- #define **MAX_COMMAND_LENGTH** 10
- #define **MAX_INPUT_LENGTH** (MAX_COMMAND_LENGTH + 100)
- #define **NOT_IN_LIST** -1
- #define **NO_EMPTY_INDEX** -1
- #define **FIELD_SIZE** 25
- #define **ADJUSTMENT_CONSTANT** 0.1
- #define **STRING_MAX_LENGTH** 10000
- #define **TABS** ' '
- #define **NOT_FOUND_STRING** " "
- #define **EDU_MAX_SUBJECTS** 10
- #define **DATABASE_PATH** "./bin/data/database.txt"

8.2.1 Detailed Description

Contains symbolic constants used throughout the program.

This header-file contains all of the symbolic constants used throughout the entire program, such as those relating to the number of regions, the max lenght of strings or constants used for string formatting.

8.3 include/database.h File Reference

Contains elements relating to the database.

```
#include "education.h"
```

Classes

- struct database

Functions

- void **freeDatabase** (struct database ∗)
- struct database ∗ **createDatabase** (char ∗)
- struct education ∗ **findEducation** (char ∗, struct database ∗)

8.3.1 Detailed Description

Contains elements relating to the database.

Contains the database struct and functions for creating, freeing and finding educations.

8.4 include/education.h File Reference

Contains elements relating to educations.

```
#include "region.h"
#include "subjects.h"
#include "vector.h"
```

Classes

- struct education

  Describes an education and all it requirements.

Functions

- struct education createDefaultEducation (int amount_of_interests, int amount_of_subjects)

  Assigns default values to the fields of the education struct.
- struct education ∗ createArrayOfEducations (int amount_of_educations)

  Allocate memory for an array of educations and return a pointer to it.
- void **freeEducation** (struct education ∗)

8.4.1 Detailed Description

Contains elements relating to educations.

This file contains the education struct and the function that creates educations.

8.4.2 Function Documentation

8.4.2.1 createArrayOfEducations()

```
struct education * createArrayOfEducations (
            int amount_of_educations )
```

Allocate memory for an array of educations and return a pointer to it.

Parameters

amount_of_educations	The amount of educations to be stored in the array

Definition at line 54 of file education.c.

```
                                                                      {
    struct education* educations;
    educations = (struct education*) calloc(amount_of_educations, sizeof(struct
   education));

    if(educations == NULL){
        printf("Failed to allocate memory for educations.\n");
        exit(EXIT_FAILURE);
    }

    return educations;
}
```

8.4.2.2 createDefaultEducation()

```
struct education createDefaultEducation (
              int amount_of_interests,
              int amount_of_subjects )
```

Assigns default values to the fields of the education struct.

Parameters

amount_of_interests	The number of interests the education should hold
amount_of_subjects	The number of subjects the education should hold

Definition at line 16 of file education.c.

```
                                                                                           {
    struct education education;
    char *temp_name = "Nothing";
    char *temp_desc = "No education selected";
    char *temp_link = "No education link";

    education.name = (char *) calloc(strlen(temp_name) + 1, sizeof(char));
    education.description = (char *) calloc(strlen(temp_desc) + 1, sizeof(char));
    education.link = (char *) calloc(strlen(temp_link) + 1, sizeof(char));
    education.region = NORTH_JUTLAND;
    education.required_grade = 0.0;
    education.interests = createVector(amount_of_interests);
    education.required_qualifications = createQualifications(amount_of_subjects);

    strcpy(education.name, temp_name);
    strcpy(education.description, temp_desc);
    strcpy(education.link, temp_link);

    return education;
}
```

8.5 include/parser.h File Reference

Contains elements relating to parsing the database.

```
#include <stdio.h>
#include <stdlib.h>
#include "database.h"
#include "region.h"
```

Functions

- void parseDatabase (struct database ∗database, FILE ∗filereader)

    Parse the database file and set all values in the database.
- void parseDatabaseLine (const char key[ ], struct database ∗database, FILE ∗filereader)

    Parse the line containing key and return into database.
- void findDatabaseLine (const char key[ ], FILE ∗filereader, char ∗current_line)

    Search the database until the first word of a line matches with key.
- int parseNumOfEdu (FILE ∗filereader)

    Returns the number of educations from database file.
- int parseNumOfInterests (FILE ∗filereader)

    Parse/count the number of intersts in the database file and return as int.
- void parseEduNames (int amount_of_educations, struct education ∗educations, char current_line[ ])

    Parses the name for each education.
- void parseEduDesc (int amount_of_educations, struct education ∗educations, char current_line[ ])

    Parses the description for each education.
- void **parseEduLink** (int amount_of_educations, struct education ∗educations, char current_line[ ])
- void parseEduRegion (int amount_of_educations, struct education ∗educations, char current_line[ ])

    Parses the region for each education.
- void parseSubReq (int amount_of_educations, struct education ∗educations, char current_line[ ])

    Parses the subject requirements for each education.
- void parseReqGrade (int amount_of_educations, struct education ∗educations, char current_line[ ])

    Parses the required average grade for each education.
- void parseInterestNames (struct database ∗database, FILE ∗filereader)

    Parse the names of each interest and return to the database.
- void parseInterestValues (int amount_of_interests, int amount_of_educations, struct education ∗educations, FILE ∗filereader)

    Parse the values for each interest in all educations and return into educations.
- char ∗ parseEduString (char ∗current_line, int amount_of_educations, int offset)

    Scans the current line + i until TABS or newline.
- char ∗∗ createArrayOfStrings (int amount_of_strings)

    Allocate memory for an array of strings and return a pointer to it.
- int **sseek** (char ∗, char)
- void readReqString (struct qualification ∗, char ∗, int)

    Read a requiremnt from a string.
- enum region strToReg (char ∗region_string)

    Converts a string into an enum region and return an enum region.

8.5.1 Detailed Description

Contains elements relating to parsing the database.

<Detailed esription="" here>="">

8.5.2 Function Documentation

8.5.2.1 createArrayOfStrings()

```
char ** createArrayOfStrings (
             int amount_of_strings )
```

Allocate memory for an array of strings and return a pointer to it.

Parameters

amount_of_strings	The amount of strings to be stored in the array

Definition at line 56 of file parser.c.

```
                                             {
    char **strings;
    strings = (char **) calloc(amount_of_strings, sizeof(char*));

    if(strings == NULL){
        printf("Failed to allocate memory for array of strings.\n");
        exit(EXIT_FAILURE);
    }

    return strings;
}
```

8.5.2.2 findDatabaseLine()

```
void findDatabaseLine (
             const char key[ ],
             FILE * filereader,
             char * current_line )
```

Search the database until the first word of a line matches with key.

Parameters

key	The term to search for
filereader	The database file
current_line	Return through this parameter

Definition at line 74 of file parser.c.

```
                                                                   {
    int found = 0;
    char temp_string[STRING_MAX_LENGTH];

    while(!found && fgets(current_line, STRING_MAX_LENGTH, filereader) != NULL){
        sscanf(current_line, "%[^\n ]s", temp_string);
        if(strcmp(temp_string, key) == 0){
            /*printf("%s WAS FOUND.........\n", key);*/
            found = 1;
        }
    }

    /* Return default string if a line with key does not exist */
    if(found == 0)
        strcpy(current_line, NOT_FOUND_STRING);

}
```

8.5.2.3 parseDatabase()

```
void parseDatabase (
            struct database * database,
            FILE * filereader )
```

Parse the database file and set all values in the database.

Parameters

database	The database to modify
filereader	The database file

Definition at line 16 of file parser.c.

```
                                                            {
    /* This will contain the first line where the type of database and encoding is read. */
    char database_format[STRING_MAX_LENGTH];
    char lines_to_read[][STRING_MAX_LENGTH] = {"NAME", "DESC", "LINK",
                                           "LOCATION", "REQUIREMENTS",
                                           "REQUIRED GRADE", "INTERESTS"};

    char amount_of_lines_to_read = sizeof(lines_to_read) / sizeof(lines_to_read[0]);
    int i;

    /* This line holds the type of database and its character encoding. */
    findDatabaseLine("EDU", filereader, database_format);

    /* Guard to make sure the file is an EDU file */
    if(strcmp(database_format, NOT_FOUND_STRING) == 0){
        printf("Error in parseDatabase: The file is not a file of format EDU.");
        return;
    }

    database->amount_of_educations = parseNumOfEdu(filereader);
    database->educations = createArrayOfEducations(database->
   amount_of_educations);

    database->amount_of_interests = parseNumOfInterests(filereader);
    database->interest_string = createArrayOfStrings(database->
   amount_of_interests);

    /* Allocate memory for interest vectors in all educations */
    for(i = 0; i < database->amount_of_educations; i++){
        database->educations[i].interests = createVector(database->
   amount_of_interests);
    }

    /* Parse all lines from lines_to_read */
    for(i = 0; i < amount_of_lines_to_read; i++){
        parseDatabaseLine(lines_to_read[i], database, filereader);
    }
}
```

8.5.2.4 parseDatabaseLine()

```
void parseDatabaseLine (
            const char key[],
            struct database * database,
            FILE * filereader )
```

Parse the line containing key and return into database.

Parameters

key	The relevant line to parse
database	The database
filereader	The database file

Definition at line 98 of file parser.c.

```
                                                                                       {
    char current_line[STRING_MAX_LENGTH];

    findDatabaseLine(key, filereader, current_line);

    /* Guard to make sure a line with key exists. If it does not, reset pointer in file and return */
    if(strcmp(current_line, NOT_FOUND_STRING) == 0){
        printf("An error has occured: Tried to parse line with %s, but entry does not exist in database.\n
\n", key);
        rewind(filereader);
        return;
    }

    if(strcmp(key, "NAME") == 0){
        parseEduNames(database->amount_of_educations, database->
educations, current_line);
    } else if(strcmp(key, "DESC") == 0){
        parseEduDesc(database->amount_of_educations, database->
educations, current_line);
    } else if(strcmp(key, "LINK") == 0){
        parseEduLink(database->amount_of_educations, database->educations, current_line);
    } else if(strcmp(key, "LOCATION") == 0){
        parseEduRegion(database->amount_of_educations, database->
educations, current_line);
    } else if(strcmp(key, "REQUIREMENTS") == 0){
        parseSubReq(database->amount_of_educations, database->
educations, current_line);
    } else if(strcmp(key, "REQUIRED GRADE") == 0){
        parseReqGrade(database->amount_of_educations, database->
educations, current_line);
    } else if(strcmp(key, "INTERESTS") == 0){
        /* First parse all the names of the interests */
        parseInterestNames(database, filereader);

        /* Rewind the file pointer and find the correct line again */
        rewind(filereader);
        findDatabaseLine(key, filereader, current_line);

        parseInterestValues(database->amount_of_interests,
                            database->amount_of_educations,
                            database->educations,
                            filereader);
    } else{
        printf("An error has occured: Attempting to parse %s, but no parsing functions exist.\n\n", key);
    }

    rewind(filereader);
}
```

8.5.2.5 parseEduDesc()

```
void parseEduDesc (
             int amount_of_educations,
             struct education * educations,
             char current_line[] )
```

Parses the description for each education.

Parses the "read further" link for each education.

Parameters

educations	An array of educations
amount_of_educations	The amount of educations
current_line	The line to parse education names

Definition at line 341 of file parser.c.

```
                                                                                    {
    int i;
    int offset = 0;

    /* Iterate through all educations */
    for(i = 0; i < amount_of_educations; i++){
        educations[i].description = parseEduString(current_line, amount_of_educations, offset
    );
        offset += strlen(educations[i].description) + 1;
    }
}
```

8.5.2.6 parseEduNames()

```
void parseEduNames (
             int amount_of_educations,
             struct education * educations,
             char current_line[] )
```

Parses the name for each education.

Parameters

educations	An array of educations
amount_of_educations	The amount of educations
current_line	The line to parse education names

Definition at line 324 of file parser.c.

```
                                                                                     {
    int i;
    int offset = 0;

    /* Iterate through all educations and assign names */
    for(i = 0; i < amount_of_educations; i++){
        educations[i].name = parseEduString(current_line, amount_of_educations, offset);
        offset += strlen(educations[i].name) + 1;
    }
}
```

8.5.2.7 parseEduRegion()

```
void parseEduRegion (
            int amount_of_educations,
            struct education * educations,
            char current_line[ ] )
```

Parses the region for each education.

Parameters

educations	An array of educations
amount_of_educations	The amount of educations
current_line	The line to parse regions from

Definition at line 256 of file parser.c.

```
                                                                                                {
    char *temp_region_string;
    int i;
    int offset = 0;

    /* Iterate through all educations */
    for(i = 0; i < amount_of_educations; i++){
        temp_region_string = parseEduString(current_line, amount_of_educations, offset);
        offset += strlen(temp_region_string) + 1;
        educations[i].region = strToReg(temp_region_string);
        free(temp_region_string);
    }
}
```

8.5.2.8 parseEduString()

```
char * parseEduString (
            char * current_line,
            int amount_of_educations,
            int offset )
```

Scans the current line + i until TABS or newline.

Saves the scanned string and returns a pointer to it.

Parameters

current_line	The line to scan
amount_of_educations	The amount of educations in database
offset	The offset to decide how many chars to skip in current_line

Definition at line 375 of file parser.c.

```
                                                                                {
    char tmp_education_string[STRING_MAX_LENGTH];
    char *education_string;

    int tmp_education_string_length;
    int i;

    /* Calculate how many chars to skip. Will always skip the first word */
    i = strchr(current_line, TABS) - current_line + sizeof(char) + offset;

    sscanf(current_line + i, "%[^\n ]s", tmp_education_string);
    tmp_education_string_length = strlen(tmp_education_string);
    education_string = (char *) malloc((tmp_education_string_length + 1) * sizeof(char));
    strcpy(education_string, tmp_education_string);

    return education_string;
}
```

8.5.2.9 parseInterestNames()

```
void parseInterestNames (
               struct database * database,
               FILE * filereader )
```

Parse the names of each interest and return to the database.

Parameters

database	The database
filereader	The database file

Definition at line 146 of file parser.c.

```
                                                                      {
    int i;
    char current_line[STRING_MAX_LENGTH];
    char temp_string[STRING_MAX_LENGTH];

    for(i = 0; i < database->amount_of_interests; i++){
        fgets(current_line, STRING_MAX_LENGTH, filereader);
        sscanf(current_line, "%[^\n ]s", temp_string);
        database->interest_string[i] = calloc(strlen(temp_string) + 1, sizeof(char));
        strcpy(database->interest_string[i], temp_string);
    }
}
```

8.5.2.10 parseInterestValues()

```
void parseInterestValues (
               int amount_of_interests,
               int amount_of_educations,
               struct education * educations,
               FILE * filereader )
```

Parse the values for each interest in all educations and return into educations.

Parameters

amount_of_interests	The amount of interests
amount_of_educations	The amount of educations
educations	The array of educations
filereader	The database file

Definition at line 184 of file parser.c.

```
                      {
    char *temp_interest_value_string;
    char current_line[STRING_MAX_LENGTH];
    int offset;
    int i;
    int j;
    struct vector temp_vector;

    /* Iterate through all interests */
    for(i = 0; i < amount_of_interests; i++){
        offset = 0;
        fgets(current_line, STRING_MAX_LENGTH, filereader);

        /* Assign the interest values to each education */
        for(j = 0; j < amount_of_educations; j++){
            temp_interest_value_string = parseEduString(current_line, amount_of_educations,
      offset);
            educations[j].interests.array[i] = strtod(temp_interest_value_string, NULL);
            offset += strlen(temp_interest_value_string) + 1;
            free(temp_interest_value_string);
        }
    }

    /* Normalize Values */
    for(i = 0; i < amount_of_educations; i++){
        temp_vector = normalizeVector(educations[i].interests);
        freeVector(educations[i].interests);
        educations[i].interests = temp_vector;
    }
}
```

8.5.2.11 parseNumOfEdu()

```
int parseNumOfEdu (
              FILE * fileReader )
```

Returns the number of educations from database file.

Parameters

filereader	The file to read from

Definition at line 296 of file parser.c.

```
                                   {
    int number_of_educations = 0;
    char current_line[STRING_MAX_LENGTH];
    int line_length;
    int i;

    findDatabaseLine("NAME", filereader, current_line);
    line_length = strlen(current_line);

    /* Iterate through all educations */
    for(i = 0; i < line_length; i++){
        if(current_line[i] == TABS) {
            number_of_educations++;
        }
    }

    /* Reset file pointer */
    rewind(filereader);

    return number_of_educations;
}
```

8.5.2.12 parseNumOfInterests()

```
int parseNumOfInterests (
            FILE * filereader )
```

Parse/count the number of intersts in the database file and return as int.

Parameters

filereader	The database file

Definition at line 163 of file parser.c.

```
                                            {
    char current_line[STRING_MAX_LENGTH];
    int number_of_interests = 0;

    findDatabaseLine("INTERESTS", filereader, current_line);

    while (fgets(current_line, STRING_MAX_LENGTH, filereader) != NULL){
        number_of_interests++;
    }

    rewind(filereader);
    return number_of_interests;
}
```

8.5.2.13 parseReqGrade()

```
void parseReqGrade (
            int amount_of_educations,
            struct education * educations,
            char current_line[] )
```

Parses the required average grade for each education.

Parameters

educations	An array of educations
amount_of_educations	The amount of educations
current_line	The line to parse the required average grade from

Definition at line 236 of file parser.c.

```
{
    char *temp_grade_string;
    int i;
    int offset = 0;

    /* Iterate through all educations */
    for(i = 0; i < amount_of_educations; i++){
        temp_grade_string = parseEduString(current_line, amount_of_educations, offset);
        offset += strlen(temp_grade_string) + 1;
        educations[i].required_grade = strtod(temp_grade_string, NULL);
        free(temp_grade_string);
    }
}
```

8.5.2.14 parseSubReq()

```
void parseSubReq (
            int amount_of_educations,
            struct education * educations,
            char current_line[ ] )
```

Parses the subject requirements for each education.

Parameters

educations	An array of educations
amount_of_educations	The amount of educations
current_line	The line to parse subject requirements from

Definition at line 220 of file parser.c.

```
{
    int i;

    for(i = 0; i < amount_of_educations; i++){
        educations[i].required_qualifications.subjects = (struct subject *) calloc(
      EDU_MAX_SUBJECTS, sizeof(struct subject));
        educations[i].required_qualifications.amount_of_subjects = 0;
        readReqString(&(educations[i].required_qualifications), current_line, i + 1);
    }
}
```

8.5.2.15 readReqString()

```
void readReqString (
            struct qualification * qualification,
            char * string,
            int education_location )
```

Read a requiremnt from a string.

Parameters

qualification	The qualification structure, where the read input is stored.
string	The string in which the requirements exists.
education_location	Which colomn is the educations requirements in.

Definition at line 420 of file parser.c.

```
                                                                                    {
    int i, subject_index=0, offset = 0, moreReqs = 1;
    char reqClass[30];

    /*Find the offset for the current education*/
    for(i = 0; i < education_location; i++)
        offset += sseek(string + offset, '\t') + 1;

    do{
        fflush(stdout);
        qualification->amount_of_subjects += 1;

        /*read the first requirement name*/
        for(i = 0; string[offset + i] != ' ' && string[offset + i] != '\n' && string[offset + i] != '=' &&
  string[offset + i] != '\t' && string[offset + i] != '_'; ++i)
            reqClass[i] = string[offset + i];

        reqClass[i] = '\0';

        qualification->subjects[subject_index].name = stringToClass(reqClass);
        if(qualification->subjects[subject_index].name == NONE){
            qualification->subjects[subject_index].name = DANISH;
            qualification->subjects[subject_index].level = Z;
        }


        if(string[offset + i] == '_'){
            ++i;
            qualification->subjects[subject_index].level =
  charToLevel(string[offset + i]);
            ++i;
        } else{
            qualification->subjects[subject_index].level = Z;
        }

        /*Check if there is more req to read*/
        if(string[offset + i] == '\t' || string[offset + i] == '\n'){
            moreReqs = 0;
        }

        ++subject_index;
        offset += sseek(&string[offset], ' ') + 1;
    } while(moreReqs);
}
```

8.5.2.16 strToReg()

```
int strToReg (
            char * region_string )
```

Converts a string into an enum region and return an enum region.

Parameters

region_string	The string to convert

Definition at line 274 of file parser.c.

```
                                            {
    enum region region;

    if(strcmp(region_string, "NORTH_JUTLAND") == 0){
        region = NORTH_JUTLAND;
    } else if(strcmp(region_string, "CENTRAL_JUTLAND") == 0){
        region = CENTRAL_JUTLAND;
    } else if(strcmp(region_string, "SOUTHERN_DENMARK") == 0){
        region = SOUTHERN_DENMARK;
    } else if(strcmp(region_string, "ZEALAND") == 0){
        region = ZEALAND;
    } else if(strcmp(region_string, "CAPITAL_REGION") == 0) {
        region = CAPITAL_REGION;
    }

    return region;
}
```

8.6 include/profile.h File Reference

Contains elements relating to user profiles.

```
#include "vector.h"
#include "subjects.h"
#include "region.h"
#include "education.h"
#include "constants.h"
```

Classes

- struct profile

  Describes a user.

Functions

- struct profile createProfile (int number_of_interests)

  Allocates memory for each of the fields in the profile struct.
- void freeProfile (struct profile p)

  Frees the allocated memory for the given profile.
- void printProfile (struct profile p)

  Prints information stored in the given profil.

8.6.1 Detailed Description

Contains elements relating to user profiles.

Contains the profile struct and the functions for creating, printing and deallocating user profiles.

8.6.2 Function Documentation

8.6.2.1 createProfile()

```
struct profile createProfile (
                int number_of_interests )
```

Allocates memory for each of the fields in the profile struct.

Parameters

number_of_interests	The number of interests allocated

Definition at line 15 of file profile.c.

```
                                                      {
    struct profile profile;
    int i;

    profile.interests = createVector(number_of_interests);
    profile.adjustment_vector = createVector(number_of_interests);
    profile.qualifications = createQualifications(TOTAL_SUBJECTS);
    profile.average = 0.0;
    profile.location.region = 0;
    profile.location.region_importance = 0;
    profile.last_recommended = 0;

    profile.adjustment_vector.array[0] = 0.0001;

    for(i = 0; i < EDUCATION_LIST_LENGTH; i++){
        strcpy(profile.saved_educations[i], "");
        strcpy(profile.recommended_educations[i], "");
    }

    for(i = 0; i < profile.qualifications.amount_of_subjects; i++)
        profile.qualifications.subjects[i].name = i;

    return profile;
}
```

8.6.2.2 freeProfile()

```
void freeProfile (
            struct profile p )
```

Frees the allocated memory for the given profile.

Parameters

p	The profile struct which is freed

Definition at line 44 of file profile.c.

```
                                  {
    freeQualifications(&p.qualifications);
    freeVector(p.interests);
    freeVector(p.adjustment_vector);
}
```

8.6.2.3 printProfile()

```
void printProfile (
            struct profile p )
```

Prints information stored in the given profil.

Parameters

p	The profile struct which is printed

Definition at line 54 of file profile.c.

```
54                                    {
55     int i;
56
57     printf("Name: %s\n", p.name);
58     printVector(p.interests);
59
60     for(i = 0; i < p.qualifications.amount_of_subjects; i++){
61         printf("%s %d\n", classNameStr(p.qualifications.subjects[i].name),
62                           p.qualifications.subjects[i].level);
63     }
64     printf("Everything is fine\n");
65
66 }
```

8.7 include/region.h File Reference

Contains geographical elements.

Classes

- struct location

Enumerations

- enum region {
  NORTH_JUTLAND = 0, **CENTRAL_JUTLAND**, **SOUTHERN_DENMARK**, **ZEALAND**,
  CAPITAL_REGION }

  Describes a region.

8.7.1 Detailed Description

Contains geographical elements.

This file contains the enums for different regions and the struct that symbolises a location.

8.7.2 Enumeration Type Documentation

8.7.2.1 region

```
enum region
```

Describes a region.

This enum descripes a region AKA it descripes a location in denmark.

Definition at line 16 of file region.h.

```
          {
     NORTH_JUTLAND = 0,
     CENTRAL_JUTLAND,
     SOUTHERN_DENMARK,
     ZEALAND,
     CAPITAL_REGION
};
```

8.8 include/subjects.h File Reference

Contains code regarding subjects and qualifcations for educations.

Classes

- struct subject
- struct qualification

Enumerations

- enum {
  MATHEMATICS, **CHEMISTRY**, **BIOLOGY**, **PHYSICS**,
  ENGLISH, **BIOTECHNOLOGY**, **GEOSCIENCE**, **HISTORY**,
  IDEA_HISTORY, **INFORMATICS**, **INTERNATIONAL_ECONOMICS**, **COMMUNICATION_AND_IT**,
  RELIGION, **SOCIALSTUDIES**, **BUSINESS_ECONOMICS**, **CONTEMPORARY_HISTORY**,
  FRENCH, **SPANISH**, **GERMAN**, **CHINESE**,
  ARABIC, **GREEK**, **ITALIAN**, **JAPANESE**,
  LATIN, **PORTUGESE**, **RUSSIAN**, **NONE**,
  DANISH }
- enum **level** { **Z**, **C**, **B**, **A** }

Functions

- struct qualification **createQualifications** (int number_of_ualifications)
- void **freeQualifications** (struct qualification ∗)
- enum stringToClass (char ∗)

  Returns the enum class associated with the given string.
- enum level charToLevel (char ch)

  Returns the enum level associated with the given char.
- char levelToChar (enum level l)

  Returns the character associated with the given enum level.

8.8.1 Detailed Description

Contains code regarding subjects and qualifcations for educations.

Contains the enums for different classes and their levels. Also includes the subject and qualification structs and some related functions

8.8.2 Function Documentation

8.8.2.1 charToLevel()

```
enum level charToLevel (
              char ch )
```

Returns the enum level associated with the given char.

Parameters

ch	The character which is converted into an enum level

Definition at line 99 of file subjects.c.

```
99                                  {
100     enum level level = Z;
101
102     if(ch == 'A')
103         level = A;
104     else if(ch == 'B')
105         level = B;
106     else if(ch == 'C')
107         level = C;
108
109     return level;
110 }
```

8.8.2.2 levelToChar()

```
enum char levelToChar (
              enum level l )
```

Returns the character associated with the given enum level.

Parameters

l	The enum level which is converted into a character

Definition at line 116 of file subjects.c.

```
                               {
    switch(l){
        case 1:
            return 'C';
        case 2:
            return 'B';
        case 3:
            return 'A';
        default:
            return 'Z';
    }
}
```

8.8.2.3 stringToClass()

```
enum class stringToClass (
            char * string )  [strong]
```

Returns the enum class associated with the given string.

Parameters

string	The string which is converted into an enum class

Definition at line 32 of file subjects.c.

```
                                        {
    if (strcmp(string, "MATHEMATICS") == 0){
        return MATHEMATICS;
    }else if(strcmp(string, "CHEMISTRY") == 0){
        return CHEMISTRY;
    }else if(strcmp(string, "BIOLOGY") == 0){
        return BIOLOGY;
    }else if(strcmp(string, "PHYSICS") == 0){
        return PHYSICS;
    }else if(strcmp(string, "ENGLISH") == 0){
        return ENGLISH;
    }else if(strcmp(string, "DANISH") == 0){
        return DANISH;
    }else if(strcmp(string, "BIOTECHNOLOGY") == 0){
        return BIOTECHNOLOGY;
    }else if(strcmp(string, "GEOSCIENCE") == 0){
        return GEOSCIENCE;
    }else if(strcmp(string, "HISTORY") == 0){
        return HISTORY;
    }else if(strcmp(string, "IDEAHISTORY") == 0){
        return IDEA_HISTORY;
    }else if(strcmp(string, "INFORMATICS") == 0){
        return INFORMATICS;
    }else if(strcmp(string, "INTERNATIONALECONOMICS") == 0){
        return INTERNATIONAL_ECONOMICS;
    }else if(strcmp(string, "COMMUNICATIONANDIT") == 0){
        return COMMUNICATION_AND_IT;
    }else if(strcmp(string, "RELIGION") == 0){
        return RELIGION;
    }else if(strcmp(string, "SOCIALSTUDIES") == 0){
        return SOCIALSTUDIES;
    }else if(strcmp(string, "BUSINESSECONOMICS") == 0){
        return BUSINESS_ECONOMICS;
    }else if(strcmp(string, "CONTEMPORARYHISTORY") == 0){
        return CONTEMPORARY_HISTORY;
    }else if(strcmp(string, "FRENCH") == 0){
        return FRENCH;
    }else if(strcmp(string, "SPANISH") == 0){
        return SPANISH;
    }else if(strcmp(string, "GERMAN") == 0){
        return GERMAN;
    }else if(strcmp(string, "CHINESE") == 0){
        return CHINESE;
    }else if(strcmp(string, "ARABIC") == 0){
        return ARABIC;
    }else if(strcmp(string, "GREEK") == 0){
        return GREEK;
    }else if(strcmp(string, "ITALIAN") == 0){
        return ITALIAN;
    }else if(strcmp(string, "JAPANESE") == 0){
        return JAPANESE;
    }else if(strcmp(string, "LATIN") == 0){
        return LATIN;
    }else if(strcmp(string, "PORTUGESE") == 0){
        return PORTUGESE;
    }else if(strcmp(string, "RUSSIAN") == 0){
        return RUSSIAN;
    }

    /*default*/
    return NONE;
}
```

8.9 include/vector.h File Reference

Contains elements relating to vectors.

Classes

- struct vector

Functions

- struct vector createVector (int size)

  creates a vector on the heap and outputs it
- struct vector copyVector (struct vector v)

  Copies the the inputted vector into vector copy and returns this.
- struct vector addVector (struct vector v1, struct vector v2)

  Adds two vectors together and outputs the sum as a vector.
- struct vector subtractVector (struct vector v1, struct vector v2)

  Subtracts the second vector from the first vector and returns the result as a vector.
- struct vector scaleVector (struct vector v, double scale)

  Multiplies the given vector's array values by the value inputted as scale, then outputs the result as a vector.
- struct vector normalizeVector (struct vector v)

  Normalises a vector via scaling it by one over it's length, then returns the normalized vector.
- double lengthOfVector (struct vector v)

  Calculates and returns the length of the given vector.
- double dotProduct (struct vector v1, struct vector v2)

  Calculates and returns the dot product of two vectors.
- void printVector (struct vector v)

  Prints a vector.
- void freeVector (struct vector v)

  frees the dynamically allocated array on the heap
- void freeVectorM (int num,...)

  Frees a variable number of struct vectors using free(Vector)

8.9.1 Detailed Description

Contains elements relating to vectors.

This file contains the vector struct and various functions used to create, manipulate or free vectors.

8.9.2 Function Documentation

8.9.2.1 addVector()

```
struct vector addVector (
              struct vector v1,
              struct vector v2 )
```

Adds two vectors together and outputs the sum as a vector.

Parameters

v1	The first vector struct: v1.array[] is a vector, v1.size number of elements in the vector
v2	The second vector struct: v2.array[] is a vector

Definition at line 83 of file vector.c.

```
                                                 {
    struct vector sum = createVector(v1.size);
    int i;

    for(i = 0; i < v1.size; i++)
        sum.array[i] = v1.array[i] + v2.array[i];

    return sum;
}
```

8.9.2.2 copyVector()

```
struct vector copyVector (
              struct vector v )
```

Copies the the inputted vector into vector copy and returns this.

Parameters

v	The input vector that is copied

Definition at line 56 of file vector.c.

```
                                      {
    struct vector copy = createVector(v.size);
    int i;

    for(i = 0; i < v.size; i++)
        copy.array[i] = v.array[i];

    return copy;
}
```

8.9.2.3 createVector()

```
struct vector createVector (
              int size )
```

creates a vector on the heap and outputs it

Parameters

size	The number of elements in the vector

Definition at line 12 of file vector.c.

```
                                       {
    struct vector vector;
    vector.array = (double*)calloc(size, sizeof(double));

    if(vector.array == NULL){
        printf("Failed to allocate memory. Bye bye.\n");
        exit(EXIT_FAILURE);
    }
    vector.size = size;

    return vector;
}
```

8.9.2.4 dotProduct()

```
double dotProduct (
              struct vector v1,
              struct vector v2 )
```

Calculates and returns the dot product of two vectors.

Parameters

v1	The first vector to be used for dot product calculation
v2	The second vector to be used for dot product calculation

Definition at line 150 of file vector.c.

```
                                                        {
```

```
    double dot_product = 0;
    int i;

    for(i = 0; i < v1.size; i++)
        dot_product += v1.array[i] * v2.array[i];

    return dot_product;
}
```

8.9.2.5 freeVector()

```
void freeVector (
              struct vector v )
```

frees the dynamically allocated array on the heap

Parameters

v	The vector struct containing the array on the heap

Definition at line 47 of file vector.c.

```
                                {
    free(v.array);
}
```

8.9.2.6 freeVectorM()

```
void freeVectorM (
              int num,
               ...  )
```

Frees a variable number of struct vectors using free(Vector)

Parameters

num	The number of arguments (vectors) that should be freed

Definition at line 29 of file vector.c.

```
                                {
    int i;
    va_list list;

    va_start(list, num);

    for(i = 0; i < num; i++){
        struct vector v = va_arg(list, struct vector);
        freeVector(v);
    }

    va_end(list);
}
```

8.9.2.7 lengthOfVector()

```
double lengthOfVector (
            struct vector v )
```

Calculates and returns the length of the given vector.

Parameters

v	The vector whose length is found

Definition at line 127 of file vector.c.

```
                                {
    double sum = 0.0;
    int i;

    for(i = 0; i < v.size; i++)
        sum += pow(v.array[i], 2);

    return sqrt(sum);
}
```

8.9.2.8 normalizeVector()

```
struct vector normalizeVector (
            struct vector v )
```

Normalises a vector via scaling it by one over it's length, then returns the normalized vector.

Parameters

v	The vector which is to be normalized

Definition at line 141 of file vector.c.

```
                                      {
    return scaleVector(v, 1 / lengthOfVector(v));
}
```

8.9.2.9 printVector()

```
void printVector (
            struct vector v )
```

Prints a vector.

Parameters

v	The vector that is printed

Definition at line 71 of file vector.c.

```
                                  {
    int i;

    for(i = 0; i < v.size; i++)
        printf("%f\n", v.array[i]);
}
```

8.9.2.10 scaleVector()

```
struct vector scaleVector (
              struct vector v,
              double scale )
```

Multiplies the given vector's array values by the value inputted as scale, then outputs the result as a vector.

Parameters

v	The vector that should be up- or downscaled
scale	The value that the vector should be scaled by

Definition at line 113 of file vector.c.

```
                                                          {
    struct vector result = createVector(v.size);
    int i;

    for(i = 0; i < v.size; i++)
        result.array[i] = v.array[i] * scale;

    return result;
}
```

8.9.2.11 subtractVector()

```
struct vector subtractVector (
              struct vector v1,
              struct vector v2 )
```

Subtracts the second vector from the first vector and returns the result as a vector.

Parameters

v1	The vector that should be subtracted from
v2	The vector that is used for subtraction

Definition at line 98 of file vector.c.

```
                                                             {
     struct vector sum = createVector(v1.size);
      int i;

      for(i = 0; i < v1.size; i++)
          sum.array[i] = v1.array[i] - v2.array[i];

      return sum;
}
```

Index