Mittuniversitetet

MID SWEDEN UNIVERSITY

DT018G, Introduction to programming in C++

Project Assignment

The final assignment where the students presents a comprehension regarding...

- … usage of strings!
- … usage of vector!
- … usage of functions!
- … usage of enumerations!
- … usage of data structures!
- … I/O operations & file handling!
- … creation of well structured program code!
- … maintain a good file structure!

Table of Contents

Project

You will create a program that is made up of many features that are placed in a number of files. The purpose of the program is to handle data on a large number of people in a list. The main program shall consist of a menu from which the user may…

- … add a person to the list
- … print the entire list of persons on the screen
- … search for a person in the list
- … delete a person from the list
- … sort the list in different ways
- … randomize the order of the list
- … save the list to specified text file
- … load list data from specified text file

In your assigned student repository you'll find the folder `Project` which's dedicated to all files relevant to this final assignment. `main.cpp` act as the program's starting point and you'll need to add necessary files in conformity to the following file structure:

```
Project/include/... // *.h
```

```
Project/src/... // *.cpp
```

Note that the build script `CMakeLists.txt` may need a reload in order to become aware of added files. In **CLion** this can be done through the menu `Tools → CMake → Reload CMake Project`

Carefully read the whole project description before starting to work on the project, and consider all details as you plan the approach!

Requirements Specification

Data

- Create a `struct` to handle the following personal data:
  … First name (`string`)
  … Last name (`string`)
  … Signature (`string`). See below for details regarding the signature!
  … Height in meters (`floating point`) with two decimals!

- Declare this `struct` in a header named `constants.h`

- Use a `std::vector` to manage the list with persons!

User input, person data

- Let the user enter the person's **first name**, **last name** and **height**!

- Let the program create a **signature** (see below)!

- Assign **first name**, **last name**, **height** and **signature** to the data structure `person` and add it to the vector!

- It should be possible to enter names like `Bo Einar von Trapp`, where `Bo Einar` would be the **first name** and `von Trapp` the **last name**!

- The program does not need to account for characters other than `a-z`, `A-Z` in regards to the names!

- **Height** should be stated with two decimals precision, but trailing zeros implicitly assumed. In other words, it should be sufficient to enter a height of `1.80 m` with the value `1.8`

Signature, three levels of difficulty

1. Signature format is `xxxyyy`, where `xxx` are the initial three letters of the **first name** and `yyy` the initial three of **last name**. The signature should only contain lowercase letters. If either **first** or **last name** consists of fewer than three letters, the signature may be shorter than six characters!

2. As in **1)** but if first or last name contains fewer than three letters the signature should be filled out with the letter `x`, so that the signature always will be six characters long. For example: `Bo Ek` becomes: `boxekx`

3. As in **2)** but the signature's uniqueness must be guaranteed. It is appropriate to use the format `xxxyyyzz` where `zz` is a sequential number!

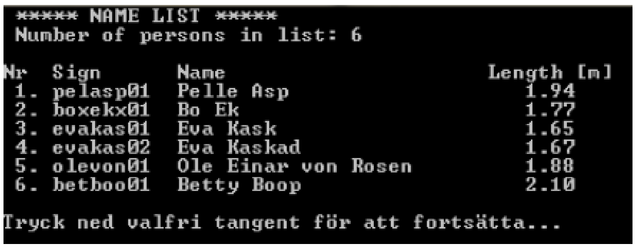
Printing information

- Use columns as demonstrated in the screenshot!

- Print a **sequential number** on each row!

- Format: **Nr**, **Signature**, **Name** and **Height**!

- **Height** is displayed with two decimals, **right aligned**!



```
 ***** NAME LIST *****
 Number of persons in list: 6

Nr  Sign       Name                      Length [m]
 1. pelasp01   Pelle Asp                      1.94
 2. boxekx01   Bo Ek                          1.77
 3. evakas01   Eva Kask                       1.65
 4. evakas02   Eva Kaskad                     1.67
 5. olevon01   Ole Einar von Rosen            1.88
 6. betboo01   Betty Boop                     2.10

Tryck ned valfri tangent för att fortsätta...
```

- Display a maximum of 20 people at a time. Allow the user to press a key to view the next 20 and next 20, and so on!

Searching / Deleting a person

- Let the user enter the person's signature as **search key**!

- **Searching:** If the requested person exists, print personal information on the screen!
- **Deleting:** if the requested person exists, remove it from the list and print information to confirm this!

- If the requested person is not in the list, print a message informing that the person is not in the list!

Sorting

- Use `std::sort` from `<algorithm>`

- Sort the list by **last name**…
  … in **ascending** alphabetical order!
  … if **last names** are equal, base the sort on **first name**!
  … The sorting should be non-case sensitive!

- Sort the list by signature, in **ascending** alphabetical order!

- Sort the list by height, in **descending** order!

- Create **one** function which will handle all sorting...
  … should be called from the menu (see below)!
  … use a parameter of `enum` type to determine which sort to perform!
  … declare this `enum` in `constants.h`

Randomized order

- Mix the persons in the list so that they appear in a random order!

File handling

- Save the list of persons to a text file!

- Write one person per line!

- Format: `first_nameDELIMlast_nameDELIMsignatureDELIMheight`

- `DELIM` is a divider represented by a character guaranteed to not be included in any name or signature (for example `|`). Declare `DELIM` as a constant in `constants.h`

- Loading data from file should replace the existing list of persons!

Main program, contents

- A `std::vector` containing persons!

- Printing the menu options (see below)!

- Calling the functions of the various menu options!

- Printing current amount of people. Print the amount of persons in connection with the printing of the menu options!

Functions

- The program should be based on functions, meaning only function calls may be made from the switch statement in `main()`

- Place function prototypes in one or several header files, `*.h`

- Place function definitions in one or several definition files, `*.cpp`

Menu

- Create a menu with the options…
  … Add a person!
  … Print the list with persons!
  … Search a person!
  … Delete a person!
  … sort by name, signature, height and randomized order!
  … Save the list to file. ***Let the user specify file name!***
  … Load data from file. ***Let the user specify file name!***
  … Quit the program!

- Place code for the menu in a function. **Input**: vector with menu options, **Output**: selected option as integer!

General

- Place main program (`main()`) in a separate file!

- Global variables are **not** allowed!

- Global constants are allowed!

- Place `enum` definitions and constants in a separate header file (`constants.h`)!

- User defined classes are **not** allowed!

- Use `C++ 11` – code! C-code, e.g. `printf()`, is **not** allowed!

Extra tasks

1. Implement level **3)** for the signature!

2. Validate uniqueness of persons. Two people are equal if **first name**, **last name** and **height** are equal. It should not matter if the names are given in lowercase or uppercase. If two people are not unique let the user choose to either **change** or **cancel** the input!

3. Encrypt data before it's written to file, and decrypt as it's read. Use displacement encryption (`rot`) and allow the user to specify the encryption key, i.e. how many steps each character should be shifted!

   NOTE!! If you have chosen to encrypt, the file format listed above does not applicable. Save an encrypted person per line and also encrypt `DELIM`!

Examination

The project gives one of grades **A**, **B**, **C**, **D**, **E**, **F** eller **F(x).** This will also be the final grade of the course!

Points for evaluation

- Specification: whether the requirements of the specification are fulfilled!

- Source code…
  … consistent code style!
  … indentation!
  … descriptive naming conventions (variables, constants, functions etc)!
  … commenting of the code!

- Structure…
  … function structure!
  … file structure!

- User-friendliness…
  … informative print-outs!
  … input of data / menu navigation!

Evaluation terminologies

Violation	*Deviations from specified requirement!*
Issue	*Inadequacies suggesting lacking comprehension of concepts. Extensive amounts of such inadequacies may (in worst case) justify a violation!*
Remark	*Weaknesses and things which could be improved. Extensive amounts of weaknesses may result in issues!*

Grade

E	Max 8 violations!
D	Max 5 violations!
C	Max 2 violations!
B	Passed for grade C and passed 2 of the extra tasks!
A	No violations and passed all three extra tasks!

NOTE!! Submissions / modifications to project repo after stated deadline will result in an automatic grade decrease (one step)!

Submission

All local changes should continuously be committed with suitable messages and it's important that these local changes are synchronized with the repo's ***remote origin*** before turning in the solution for evaluation. When this is done, you need to formally submit the work in Moodle by using the dedicated submission box. You should not attach any files to this formal submission, but need to write a small comment (such as a link to the repo)!

The program code needs to be well structured and have a layout in accordance to common conventions. This concerns *tabbing and white spaces*, *describing name conventions*, *extra line-breaks* and *suitable comments* to support readability!

The work needs to be individually conducted, but you are encouraged to discuss and support each other using the course's communication platform!