# Lab 3 in C++ OOP

Hergeir Winther Lognberg
Hewi1600

## 1 Preamble

The Lab contains folders, each containing an header and corresponding cpp file. The names of the folders correspond to the classes they contain. In the Lab folder itself we find the *Functions.cpp* and *Functions.h* Containing some functions who had no business being in the *U1.cpp* file. Such as *toInt*, *edgeTrim*, *nameFormat* and so forth.

### 1.1 Compilation

For compiling i use g++ and created a makefile to make the compiling process easier. Just run *"make"* within the lab directory compile.

## 2 The Software

I've tried to keep redundancy away from my code. Each class is only written once and reused in this lab.

Also I have tried to use reference and *const* everywhere it made sense to do so.

The only get member-function where I didn't use *const* was *getPerson* where I wanted to change the Person using it's member-functions, so *const* did not make sense in this case.

### 2.1 U1

*U1* contains nothing but a main function to create an object of the class *Userinterface* and calling its only public memberfunction *run()*.

The functions *getInt* and *getLine* are gold. Both ensure that user input can't be empty. Added to that both keep on asking for input as long as input is invalid. Lastly *getLine* ensures that an input only containing spaces will be perceived as empty.

Would love any feedback on good code practice.

### 2.2 comments

I know that overloading the operators *"«"* and *"»"* in PersonList class wasn't a requirement. But had already done so when I found out. That's the reason why I used the *this* pointer in the *readFromFile* and *writeToFile* memberfunctions of *PersonListClass*. I know how I could have done without It (Just iterate over the overloaded operators in *PersonClass*), just didn't want to rewrite any code.

## 3 Enviroment

I'm programming on an Arch Linux 64-bit system. I've got the c++ compiler installed and compile using it's g++ alias which links necessary libraries automatically. To compile i use the recommended flags: "-std=c++11 -Wall -pedantic". The flags let me choose to use c++11 standard and give me useful compiling warnings and errors.

Sunday 5$^{th}$ February, 2017