# Excercise set week five

Feedback on these excercises is very much appreciated. Send mail to one of the lecturers or to Aryan at aryannm@gmail.com

Solutions are available in file `solutions.md`.

## Problem X: (Repetition)

Write a function that takes an array of integers as parameter with void return type that increments every value of the array by 2 using both `*` dereference operator and `[]` index operator.

## Problem One - Function pointers

1. Create a function called `cynic_estimate` that takes an integer parameter and returns it mulitplied by 1. Use this function by name and by a pointer to the function (refer to lecture notes or google on how to capture function pointers)
2. Create a function called `reasonable_estimate` that does the same as above, except it multiplies by 2, use it by pointer and by name.
3. Create a function called `optimistic_estimate` that does the same above except now it multiplies by 5. Use this function by pointer and by name.
4. The keyword auto can be used to make the left-hand-side of an expression much cleaner in C++. Eg the following `int(*cynic)(int) = &cynic_estimate;` can be replaced with `auto cynic = &cynic_estimate;`. Use auto for the 3 problems above.
5. (Extra) Now you are wondering why we had to do all of that. Well suppose you have a function called `economic_forecast` that returns a different forecast depend on what function it has to use to estimate future values. Create this function `int* economic_forecast(int* array, int(*fct)(int))` and use it to calculate the forecast of next week (which is an array of 7 values) given the earnings of this week (stored in an array of 7 values) using the different estimate functions you created. (Hint, you have to allocate an array of 7 ints inside that function, what happens if you dont?)

This style of passing around function pointers is very C-like and is widely used in old libraries that use a lot of math, you should try to avoid this, you have modern C++ at your fingertips! ##Problem Two - New / Delete & bad_alloc Refer to lecture for details. 1. You have learned that you can allocate dynamic memory using the `new` keyword, but suppose there is nothing left to allocate or allocation fails for whatever reason. What happens then? 2. Can you create a program that will fail at allocation? 3. Everytime we allocate dynamic memory (whenever we use new) we have to clean up the memory we allocated when it is

no longer used. Create a program where you demonstrate this. What happens if we don't clean up? ##Problem Three - Structs 1. Create a struct called `Person` that has 3 member variables, name, lastname and age, of type string, string and int respectively. 2. Create a function inside the `Person` struct called `print()` that prints out name,lastname and age of this person. Eg with `c++ Person aryan{"Aryan", "Naqid", 28}; aryan.print();` The output should be `Hi, my name is: Aryan Naqid, age 28`. 3. So far so good, but this time make the name,lastname and age members private in the struct. What happens then with the program you already had? Does it compile? Why not? How can you fix this? (Hint: constructor) 4. What is the point of making something private, or public? 5. Create functions inside your struct for getting and setting the variables name, lastname and age. Eg for the age `void setAge(int age)` and `int getAge()`. 6. Create a Person dynamically, using `new`. Try to change the name and age of this person. (Hint, pointer to Person) 7. Find out how many bytes in memory a Person takes.

What you have done sofar is called encapsulation, which gives you as designer of the struct control over how data is accessed and hiding how the data is actually implemented. ##Problem Four - Classes Create a class Person that has member data `name`, `lastname`, `age` of type `std::string`, `std::string`, `int` respectively 1. Create a Person and set its `name` member to `"Foo"`. What happens? (Hint access modifier) Now that we know that if we don't specifyc an access specifier the default for class is used (which is private vs public for struct) so we can't access name directly.

2. Create an appropiate constructor for person. Create setter and getter functions like you did with the struct.
3. Add a new datamember for the person called `parent` of type `Person*` eg: "'c++ class Person { Person* parent; std::string name; ....omitted..

} "' Now try creating a person and setting its parent. 4. (Extra) with the class you have above and the information on https://en.wikipedia.org/wiki/Kings_of_Norway_family_tree try to create a male line of heritage for Harald V since Oscar the first (1799-1859)

## Problem Five - Tidy code

Before we go ahead it might be of value to read the document `http://www.learncpp.com/cpp-tutorial/89-cla` especially the section titled `Putting class definitions in a header file`. This is useful knowledge for keeping code structured and clean.

1. Read the document linked above.
2. Create the person class from the previous problems using header file `person.h` and cpp file `person.cpp` ##Problem Six - Composition Refer to lecture for more details.
3. Create a class called `Wheel` which has data members `type` and `diamaeter` decide if they should be int/string etc for yourself. Create an apropiate constructor. Create setter and getter for both type and diameter.

4. Create a class called `Car` which has data members `model`, `milage` and either an array of `Wheel`s or a `std::vector<Wheel>` of wheels. Which one do you think is better? The car should only have 4 wheels, and index 0 shuld be frontleft wheel, index 1 should be frontright, index 2 backleft, index 3 backright.
5. Create functions for setting and getting the model and mileage of this car.
6. Create different functions for setting and getting either the frontleft, frontright, bacleft, backright wheel of this car.
7. Create a member function that lets you change the type of the frontleft car.

The example above is a bit contrived, but composition truly shines when you have a bit larger project. The point of composition is to delegate a problem (such as changing the type of a wheel in a car) to another class without showing how it is done to the caller (eg when I call `changeTypeOfFrontLeft` I have no idea how Car decides to change it, I just know it is being changed).