Exercise 4.1 – Initialization order

- The goal of this exercise is to learn how to analyze the 'has-a' relations between a number of classes
- Approach – Creating a single Telephone object
  a) Examine the C++ source code in directory ex4.1/ and draw a tree diagram with a box for each class and a line for each has-a relationship between two classes (hand in the drawing!).
  b) Write a small main program that instantiates a `Telephone` object. You will see that your program does not compile due to an error in one of the source files from ex4.1/. Explain the error and fix the problem.
  c) Once the program is compiling and running OK, you will see that you get a printed message for each time a class constructor or destructor is called. Explain the order in which the constructor and destructors are called according to your program output.
- Approach – Copying a Telephone object
  d) Update the main program to make a second `Telephone` object that is copied from the first one using the copy constructor output.
  e) Compile and run the program and look at the constructor and destructor messages that originate from the `Telephone` copy constructor call. Does it look OK to you?

Exercise 4.1 – Initialization order

f) Examine the code of class `Telephone`, find the problem and fix it. Run the program again and convince yourself that the copy constructor works OK now.

- Approach – Using dynamic memory allocation

g) Now we will change class `Dialer` so that it owns a dynamically allocated array of `Button`s allocated through `new[]` in the constructor rather than a statically sized array of `Button`s.

h) Modify the constructor, destructor and copy constructor to implement this change.

Exercise 4.2 – The calorimeter example

- Goal: Write the classes that implement the calorimeter example of this module the course, i.e. the classes
  - `Point`
  - `CaloCell`
  - `CaloGrid`
  - `Calorimeter`

- Approach - For each of these 4 classes, *first write the interface (class declaration)*, then the implementation.
  - General hints before you start:
    - *Write a small main program* that you can use to instantiate the classes you write so that you can debug them
    - Write the classes *in the order as listed in the next pages*
  - Detailed specifications for each class follow

Exercise 4.2 – The calorimeter example

- Class `CaloCell`
  a) Supply the data members `double energy` and `int ID`
  b) The constructor should take as arguments the initial `energy` and the `ID`. Do you need a copy constructor? Do you need a Destructor?
  c) Write accessors and modifiers for data members `energy, ID`
  d) Think about which member functions should be `const` and make those const
  e) Before proceeding to the next class, test your code by creating and using a `CaloCell` object in `main()`

- Class `Point`
  f) Supply the data members `double x,y,z`
  g) The constructor should take as arguments the initial values of `x,y,z` with default of (0,0,0). Do you need a copy constructor? Do you need a destructor?
  h) Write accessors and modifiers for data members `x,y,z`
  i) Also add a modifier that sets `x,y,z` in a single call.
  j) Think about which member functions should be `const` and make those `const`
  k) Before proceeding to the next class test your code by creating and using a `Point` object in `main()`

Exercise 4.2 – The calorimeter example

- Class `CaloGrid`
  - l) Supply the data members `int nx, ny` to hold the grid dimensions and a one-dimensional array of `CaloCell` elements
  - m) Write a constructor that takes the size in x and y as arguments. Write a copy constructor and destructor.
  - n) Test your code at this point. You will probably get a compiler error about the instantiation of an array of `CaloCells` objects. What are the requirement on the constructor(s) of a class if you want to be able to allocate it as an array? Modify the `CaloCell` constructor to make it work.
  - o) Add a non-const version of `CaloCell* cell(int x, int y)` that returns a cell for a given coordinate. Check that the given `x` and `y` values are inside the range of your calorimeter grid. If they are outside return a null pointer.
  - p) Add a function `const CaloCell* cell(int x, int y) const.`
    Optional: do you need to duplicate the code of the non-const `cell()` function? (Hint: think about using `const_cast<>` and `this`; a technical discussion of typecasting can be found on http://www.cplusplus.com/doc/tutorial/typecasting/)
  - q) Test your code again
- Class `Calorimeter`
  - r) Supply two data members: a `CaloGrid` and a `Point`
  - s) Add a constructor that takes the size of calorimeter and the initial position as arguments and pass those values to the CaloGrid and Point data members. Provide a default value for the position values
  - t) Add the following accessors and modifiers: `CaloGrid& grid(), const CaloGrid& grid() const, Point& position()` and `const Point& position() const`
  - u) Do you need a copy constructor? Do you need a destructor?

Exercise 4.3 – Operator overloading

- The goal of this exercise is to use operator overloading to make class String behave more naturally
  - Copy the file in directory `ex4.3/`, which contains an initial implementation of `class String`
- Approach – Adding an assignment operator
  a) Make a little main program in which you perform a simple exercise with `class String`, for example make an instance, ask for its length and print its contents.
  b) Now add an *assignment operator* as member function to `class String`. Think first about the declaration of the operator function. What should the return type of `operator=()` be?
  c) Next, implement the body of `operator=()`. Use the private helper function `insert()` to accomplish your goal.
  d) Modify your test program so that it uses the assignment operator: assign one string to another. Also test if 'chain assignment' works (a = b = c)

Exercise 4.3 – Operator overloading

- Approach – add the addition operators `operator+=()` & `operator+()`

  e) Write first the *declarations* of these operators. What should the return types of `operator+=()` and `operator+()` be? Should `operator+()` be a member function of class String or be a global function?

  f) Write the body of `operator+=()` first. First allocate a buffer that can hold the combined string using `new[]`, then copy both strings to the new buffer using `strcpy` and finally delete the old buffer and install the new buffer. Do not forget to return value!

  g) Write the body of `operator+()` by using `operator+=()`. Hint: Copy the 1st argument of `operator+()` then use `operator+=()` to append the 2nd argument to it.

  h) Modify your test program so that it uses both the + and += operators and verify that all works OK.

  i) Can you add a string literal to a `String` using `operator+()`, as shown below? Explain why this is/isn't possible.

```
String s("Blah") ;
s += "Blah" ;
```

Exercise 4.3 – Operator overloading

- Approach – Making `class String` 'backward compatible'
  - j) It is nice if you can use class String where ever a 'const char*' is expected so that you almost never have to deal with old fashioned character strings again, even if Standard Library or other functions expect them as input argument. You can achieve this if C++ can perform an automatic conversion from String to const char* for you whenever that is required. To enable C++ to do this you must implement the appropriate 'conversion operator'
  - k) Implement a conversion operator to `const char*`. First think about the declaration of such an operator. Remember that conversion operators in general take the form '`operator TYPE() const`', where TYPE is the type you are converting to. Implement the operator function as well. Hint: it takes only 1 line of code!
  - l) Try the conversion operator by passing a `String` to Standard Library function `strlen()` (declared in `<cstring>`)

- Approach (optional) - Implement a substring function by implementing `String operator(int start,int stop)`
  - m) Don't forget to check the input and output values for validity. What can you do if they are out of range?
  - n) Why can't you use `operator[int start,int stop]` for this?