Exercise 2.1 – Sorting numbers and strings

- The goal of this exercise is to learn how to use pointers and references with functions
  - We will write a program that sorts an array of 10 random integers using the bubble sort algorithm.

```
for (i=0; i<n-1; i++) {
  for (j=0; j<n-1-i; j++) {
    // if A[j+1]>A[j] swap A[j] and A[j+1]
  }
}
```

- Approach – integer sorting
  a) Write a small main program that allocates an array of 10 integers and fill them with random values.
     a) You can manually provide a series of 10 'random' numbers in the initializer of the array in your code. You do not need to develop code that calls a random number generator and fills the array.
  b) Add the above bubble sort algorithm in a separate `sort()` function that takes the array of integers as input.
  c) Provide the missing piece of the sort algorithm: an `order()` subroutine that takes *references* to two `ints` and that swaps their values if the 2nd argument is greater than the 1st argument.
     a) NB: Note that the `order()` routine you're asked to develop here has a different functionality than the `swap()` routine in the course material of section 2
  d) Add code to your main program that prints out the array after sorting to verify that all works correctly.
  e) Now reimplement the `order()` function using *pointers* and adjust `sort()` accordingly. Do you like the pointer or the reference version better?

Exercise 2.1 – Sorting numbers and strings

- Approach – string sorting

  f) Make a copy of the main program and change it so that it allocates an array of 10 `const char*` pointers and initialize them with 10 string literals (e.g. "blah"). Why do you need the `const` here?

  g) Adjust the arguments of the `sort()` and `order()` functions to accept the array. What type of argument should `order()` take? The easiest way to figure it out is to think of '`const char*`' as a fundamental type and proceed as usual

  h) Adjust the contents of `sort()`. You can use the `strcmp(a,b)` function declared in `<cstring>` from the standard library to compare the strings. This function returns an integer value greater or smaller than zero depending on the lexical order of the two input strings. Why can't you just compare the pointer values to compare the strings?

  i) Reimplement the `order()` function using references to pointers

Exercise 2.2 – Function overloading

- The goal of this exercise is to understand the basics of function overloading.
- Approach

  a) Implement the following overloaded `min()` functions

  `min(int,int)`, `min(double,double)`, `min(int[],int)`

  where the last function returns the minimum of an array of integers with a length specified by the 2nd argument

  b) Write a small program that tests your three implementations

  c) Now try call `min()` passing a `double` and an `int` as argument. Why doesn't this work?

  d) Fix this problem without using explicit casts

Exercise 2.3 – Namespaces and scope

- The goal of this exercise is to understand the scoping rules of name spaces
- Approach: examine the following namespace definition
  a) Identify which `print()` functions are called in `sub1()` and `print()` and explain why?
  b) Is the statement '`using Black::print`' legal? Explain why?

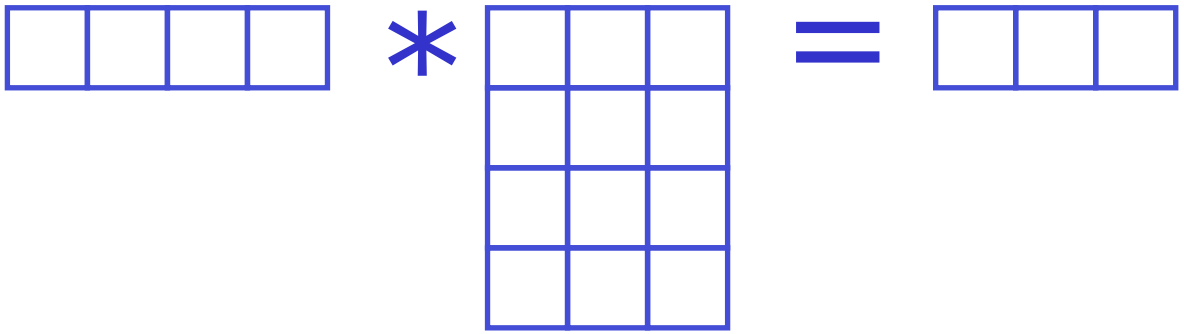
```
namespace Black {
  void print(int k) {} ;
}

namespace White {
  void print(int k) {} ;
}

// b) Global using declaration -- OK?
using Black::print ;

void sub1() {
  using White::print ; // Local using declaration
  print(5) ;           // a) Which print() is called?
}

void print(int k) {
  if (k>0) {
    print(k-1) ;       // a) Which print() is called?
  }
}
```

Exercise 2.4 – Library exercise

- The goal of this exercise is to learn the techniques involved in packaging your code as a library in a modular and usable way
- Approach - Create a library with overloaded min() and max() routines for general use
  a) Move the overloaded `min()` functions from exercise 2.2 into `min.cc`
  b) Make a file `max.cc` with corresponding `max()` functions
  c) Move the declarations of the min and max functions into separate files: `min.hh` and `max.hh` respectively
  d) Compile `min.cc` and `max.cc` into *object files* and collect them into a library `libMinMax.a`. Protect your header files against multiple inclusion and test this.
  e) Write a small test program that uses `min()` and `max()` from the library and compile and link the test program with `libMinMax.a`
  f) Define a new function int `min(int,int)` *in your test program* and compile and link your program again with `libMinMax.a`. Explain the errors you get
  g) Now modify `min.cc, min.hh, max.cc` and `max.hh` to move the declarations and definitions of all functions into a namespace `mylib`. Recompile your library
  h) Adapt the your test program to use both the `min(int,int)` function defined in the library and the version defined inside the test program.

Exercise 2.5 – Passing multidimensional arrays

- The goal of the exercise is to learn how to use multi-dimensional arrays in C++.
- Approach – Write a function that multiples a vector of length N with a matrix of size N * 3



```
double* multiply(double ivec[], int N, double mtx[][3])
```

a) First allocate the return value array using the new[] operator

b) Implement the multiplication loop.

c) Write a program to test your function. In this program initialize your matrix using the `double[n][m] = { … }` initializer. Try to understand how the one-dimensional list of initializer elements maps on the two-dimensional array