Exercise 1.1

1. `int main(int argc, char** argv) {}`
   Passes command line parameters to the main method. 'argc' stands for 'argument count' and specifies the number of parameters as an integer. If two arguments are passed, the argument count = 3.

   'argv' stands for 'argument vector' and contains the passed contents in an "array". Since strings are also represented as arrays, it is so to speak an array of strings or an array of an array. Therefore the double pointer.

   An alternative way to write this line would be: `int main(int argc, char* argv[]) {}`

2. `float foo(int x){}`
   Code compiles with the warning that no value is returned in a non-void function. It should return a float number but it doesn't.

   Also if this line of code is meant to define 'foo' it should do something in the function body and have a return value.

   And if this line of code is meant to declare 'foo', then the curly brackets are not needed.

   As it is now. This function does nothing but prompts a warning.

3. `int foo(double result=10.5){return result;}`
   Code compiles just fine. However it's poorly written or misleading because the return type is declared as int but then a double is casted to an int.
   So given 3.5 it will only return 3. 'foo' also has 10.5 set as standard value for the result. so when 'foo' is called without any parameter, it will return 10. Which is the integer part of 10.5.

4. `int foo(){return int z;}`
   This does not compile. 'int z' is only a declaration. At this point 'z' is not defined and can therefore not be returned.

5. `typdef struct{int x,y,z;}a;`
   This does not compile because it has a typo. Actually the same typo as on slide 85 from the C++ Einführung.
   If written correctly: `typedef struct{int x,y,z;}a;`
   it would create an alias 'a' for the given struct and we could write something like this:

```
1 typedef struct{int x,y,z;}a;
2
3 a my_object;
4 my_object.x = 1;
5 my_object.y = 2;
6 my_object.z = 3;
```

Exercise 1.2

(a) There are 4 types of mistakes:

1. "wrong language"
   In c++ the syntax is `#include` and not `#import` as it is in java and others.
2. "mismatched return values"
   Return value of `calculate(..);` is declared as float but defined as double. Make both double.
   Return value of `construct_bmi(..)` is declared as BMI but not defined at all. Make both BMI.
3. "no namespace"
   `cout` or `cin` are unknown. Either write `std::cout` and `std::cin` or add the entire namespace at the beginning with `using namespace std`.
4. `bmi.cpp` and `main.cpp` both include `bmi.h`-functions and those need to be called with `bmi::construct_bmi(..)` and `bmi::calculate(..)`. (or set a new namespace)

(b) The terminal commands to compile `main.cpp, bmi.cpp, bmi.h` are:
`g++ -c main.cpp` to compile the main file but not link it (-c).
`g++ -c bmi.cpp` to compile the bmi file but also not link it (-c).
`g++ main.o bmi.o -o bmiprogram` to link the first two files together and create a executable file called "bmiprogram". It's important that the file with the main() function is listed first in this command!

Exercise 1.3

q3.cpp

```cpp
#include <iostream>
#include <string>

// reverses the given String by iterating over it from right to left
// and appending every character to sf. (c)
std::string reverse(std::string s) {
    std::string sf;
    for (int i = s.length()-1; i >= 0; i--) {
        sf.append(1, s[i]);
    }
    return sf;
}

// input strings are given via command line arguments.
int main(int argc, char* argv[]) {
    std::string sa = argv[1]; // Parse arguments
    std::string sb = argv[2];
    std::string sc, sr;
    // Check if two strings were passed. Return if not.
    if (argc != 3) {
        std::cout << "Please provide two params";
        return 0;
    }

    // for every char in String a, iterate through string b and check if
    char exists.
    for (int i = 0; i < sa.length(); i++) {
        // if char i from string a is not found in b , add it to sc. (a)
        if (!(sb.find(sa[i]) != std::string::npos)) {
            sc.append(1, sa[i]);
        }
    }

    for (int i = 0; i < sb.length(); i++) { // Vice versa (b)
        if (!(sa.find(sb[i]) != std::string::npos)) {
            sr.append(1, sb[i]);
        }
    }
    std::cout << "String a: " << sa << << "String b: " << sb "\n";
    std::cout << "a - b = " << sc << "\n"; // exercise a
    std::cout << "b - a = " << sr << "\n"; // exercise b
    sr = reverse(sc);
    std::cout << sc << " reversed: " << sr << "\n"; // exercise c
    return 0; // Tell OS that program finished successfully.
}
```

Exercise 1.4

q4.cpp

```cpp
#include <iostream>

typedef struct{double x, y;} vec2d; // define vector 2d

typedef struct{double a, b, c, d;} mat2d; // define matrix 2d

// solver function takes a vector and a matrix
// and multiplicates them together.
vec2d solve(vec2d v, mat2d m) {

    vec2d u;

    // Transform matrix with gauss algorithm
    m.c = m.c - m.a * (m.c / m.a);  // m.c should now be 0
    m.d = m.d - m.b * (m.c / m.a);
    v.y = v.y - v.x * (m.c / m.a);
    u.y = v.y / m.d;
    u.x = (v.x - (m.b * u.y))/m.a;
    return u; // return vector u (solution)
}

int main() {
    vec2d v; // create example vector
    v.x = 2;
    v.y = 1;
    mat2d m; // create example matrix
    m.a = 1;
    m.b = 1;
    m.c = 0;
    m.d = 5;
    vec2d w = solve(v, m); // solve

    // should print out the vector (11/5)
    std::cout << w.x << std::endl << w.y << std::endl;

    return 0;
}
```

Exercise 1.5

q5.cpp

```cpp
#include <iostream>

enum operations { // enum containing all supported operations
    addition,
    subtraction,
    multiplication,
    division,
    modulo
};

double calc(operations op, int a, int b) {
    switch (op) {   // does the given operation
        case addition: return a+b;
        case subtraction: return a-b;
        case multiplication: return a*b;
        case division: return a/b;
        case modulo: return a%b;
    }
}

union test {int x; double y;}; // example union for 5c

int main() {
    int a = 10;
    int b = 5;
    std::cout << "10 + 5 = " << calc(addition, a, b) << std::endl;
    std::cout << "10 - 5 = " << calc(subtraction, a, b) << std::endl;
    std::cout << "10 * 5 = " << calc(multiplication, a, b) << std::endl;
    std::cout << "10 / 5 = " << calc(division, a, b) << std::endl;
    std::cout << "10 % 5 = " << calc(modulo, a, b) << std::endl;

    test q; // example to show how one value of a union overwrites another.
    q.x = 1;
    q.y = 2.5;
    std::cout << "q.x " << q.x << " y: " << q.y << std::endl;

    return 0;
}
```

(c) A union is basically a struct with the exception that all members start at the same memory address. So unlike with structs, in a union usually only one member is used. A usecase could be to have a tree structure where each knot can be a double or an int. However its usually never both. Because all memebrs start at the same address, if we define multiple members they get overwritten.

Exercise 1.6

(a) **q6a.cpp**

```
1  #include <iostream>
2  void foo (int *ptr) {
3      int *a = 5; // can't assign an int to a pointer
4      *ptr = *a; // can't derefence a variable
5  }
6
7  int main() {
8      int b = 7;
9      int *ptr;
10     *ptr = b; // runtime error: can't derefence a non-initialized
       pointer.
11     foo(ptr);
12
13     std::cout << *ptr << std::endl;
14     return 0;
15 }
```

The (two) main mistakes are on line 3 and 4. First we're trying to assign an integer to a pointer and then we try to dereference that integer. To fix it, just delete line 3 and directly write `5` on line 4 instead of `*a`. Then the runtime error is on line 9 and 10 where a pointer is declared but not initialized. Still we try to dereference that pointer. To fix that, just delete line 10 and define the pointer directly.

A working solution looks like this:
q6a-solved.cpp

```
1  #include <iostream>
2
3  void foo (int *ptr) {
4      *ptr = 5; // dereferenced ptr holds value 7 which is changed to 5
5  }
6
7  int main() {
8      int b = 7; // define int b
9      int *ptr = &b; // pointer points to address of b
10     foo(ptr);   // pass pointer to foo
11
12     std::cout << *ptr << std::endl;
13     return 0;
14 }
```

(b) The function should return a pointer pointing at the beginning of the new array. However we then don't have the length of the new array which makes it cumbersome to work with it. I therefore created a struct that includes both the pointer to the array and the length of the array. So the function `combineReverseEvenWithLength(..)` returns both values at once.
Because the Question explicitly asks for a function that "returns the pointer to the beginning of the new array" I also added an additional function `combineReverseEven(..)`to just return the pointer.
This way I can easily test my implementation with the `combineReverseEvenWithLength(..)` but still offer the requested functionality with the `combineReverseEven(..)` function.

q6b.cpp

```
#include <iostream>

typedef struct{ // Combine array length with array
    int *ptr;
    int len;
}arr;

// Take even numbers of reversed b and add even numbers of a.
// Return pointer to new array and length.
arr combineReverseEvenWithLength(arr a, arr b) {
    // Count even Numbers on both arrays
    int numOfEvenElem = 0;
    for (int i = 0; i < a.len; i++) {
        if ((*(a.ptr+i) % 2) == 0) { numOfEvenElem += 1; }
    }
    for (int i = 0; i < b.len; i++) {
        if ((*(b.ptr+i) % 2) == 0) { numOfEvenElem += 1; }
    }

    arr res; // Create new Array to fit all the even numbers of a and b
    res.ptr = new int[numOfEvenElem];
    res.len = numOfEvenElem;

    int c = 0; // Reverse the second array and add even numbers to res
    for (int i = 1; i < b.len+1; i++) {
        if ((*(b.ptr+b.len-i) % 2) == 0) {
            *(res.ptr+c) = *(b.ptr+b.len-i);
            c += 1;
        }
    }
    // Add even Numbers of first array to res
    for (int i = 0; i < a.len; i++) {
        if ((*(a.ptr+i) % 2) == 0) {
            *(res.ptr+c) = *(a.ptr+i);
            c += 1;
        }
    }
    return res;
}
```

```
40
41 //combines arrays into struct and calls other function to solve.
42 int* combineReverseEven(int* a1, int* b1) {
43     arr a; a.ptr = a1; a.len = 3;
44     arr b; b.ptr = b1; b.len = 5;
45     arr res = combineReverseEvenWithLength(a, b);
46     return res.ptr;
47 }
48
49 int main() {
50     // Initialize test arrays
51     int a1[] = {1, 2, 4};
52     int b1[] = {4, 5, 6, 8, 11};
53
54     // combine into arr struct
55     arr a; a.ptr = a1; a.len = 3;
56     arr b; b.ptr = b1; b.len = 5;
57     // calculate
58     arr res = combineReverseEvenWithLength(a, b);
59
60     // Print array to check results
61     for (int i = 0; i < res.len; i++) {
62         std::cout << *(res.ptr + i);
63     }
64     std::cout << std::endl;
65
66     return 0;
67 }
```

(c) This code does not compile. p stores an address which can't be multiplied (line 13) or subtracted (line 14). It doesn't make a difference if we write `(p-p-8)` or `(p+(p-8)` as neither subtraction nor addition of two pointers is allowed.
Pointers can be incremented or decremented by integers and can be compared. It also wouldn't make any sense to add, subtract, multiply or divide two pointers as there would be no way to tell what is stored at the resulting address.

q6c.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int a = 4;
7      int b = a + 6;
8      int *p = &a, **q = &p;
9      *p *= 4**&b***q;  // Never write such minified lines in your code!
10     p = &b;
11     *p = a + 5;
12     (*p)++;
13     p *= 2; cannot multiply pointers or addresses
14     cout << a << " " << b << " " << p << " " << (p - p - 8)) << endl; //
       cannot substract pointers either.
15
16     return 0;
17 }
```

Exercise 1.7

As you can see in line 23, the `compare(..)` function receives a function pointer as parameter. The syntax for that is `[return value] (function pointer) (param1, param2, ...)`. Obviously the number of parameters have to match. The `compare(..)` function then does nothing else but call the passed function and return its return value.

q7.cpp

```
1  #include <iostream>
2  #include <math.h>
3
4  // Compares two double numbers
5  int compareDouble(double d1, double d2) {
6      if (d1 == d2) {return 0;}
7      return (d1 > d2) ? 1 : -1;
8  }
9
10 // compares according to the sin value of the second element in the array.
11 int compare_sin(double* arr1, double* arr2) {
12     return compareDouble(sin(arr1[1]), sin(arr2[1]));
13 }
14
15 // compares according to their respective L1 values.
16 int compare_taxicab(double* arr1, double* arr2) {
17     double manh_d1 = sqrt(arr1[0] * arr1[1] * arr2[2]);
18     double manh_d2 = sqrt(arr2[0] * arr2[1] * arr2[2]);
19     return compareDouble(manh_d1, manh_d2);
20 }
21
22 // main compare function
23 int compare(double* arr1, double* arr2, int (*fptr)(double*, double*)) {
24     return fptr(arr1, arr2);
25 }
26
27 int main() {
28     double arr2[] = {75, 5, 29};
29     double arr1[] = {1, 6, 7};
30
31     std::cout << compare(arr1, arr2, compare_sin) << std::endl;
32     std::cout << compare(arr1, arr2, compare_taxicab) << std::endl;
33 }
```

Exercise 1.8

q8.py

```
1 import sys # That is a lot shorter than the mess in 1.3
2 print(set(sys.argv[1]).symmetric_difference(sys.argv[2]))
3 print(max(sys.argv[1::], key=len), len(max(sys.argv[1::], key=len)))
```