Exercise 1 - Nico Bachmann

Question 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.

You could also write the line "directly" as an array:

```
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. Here it should return a float number but it doesn't.

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

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. 'foo' receives a double and returns an integer. 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. Declaring 'z' before the return would work and return 1

5.

```
        typdef struct{int x,y,z;}a;
```

This does not compile because it has a typo.

Correct:

```
        typedef struct{int x,y,z;}a;
```

This instead would create an alias 'a' for the 'struct{int x, y, z}'.

You could now write something like:

```
        a my_object;
        my_object.x = 1;
        my_object.y = 2;
        my_object.z = 3;
```

Question 2

Error 1: wrong language

- "#include" instead of “#import" in the main.cpp

Error 2:

- "using namespace std" is missing or "cout","cin" have a missing "std::" in main.cpp:

Error 3: wrong return values

- return value of "calculate" declared as float but defined as double. Should be double.
- return value of construct_bmi declared as BMI but not defined at all. Should be BMI.

Error 4: bmi.h included twice.

- bmi.cpp and main.cpp both include bmi.h. but main also includes bmi.cpp. Every Header should have "#pragma once" in it to avoid multiple insertions

Question 5c

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 never both.

Because all memebrs start at the same address, if we define multiple members they get overwritten.

Question 6 a

Incorrect implementation:

```
#include <iostream>

void foo (int *ptr) { // here a copy instead of a reference to ptr is
made.
        int *a = 5; // Error: a pointer needs to point to an address. not
an integer
        *ptr = *a; } // We want to rewrite the pointer itself


int main() {
```

```
        int b = 7;
        int *ptr;
        *ptr = b;

        foo(ptr);

        std::cout << *ptr << std::endl;
        return 0;

}
```

Correct implementation:

```
#include <iostream>

void foo (int *&ptr) { // now we reference ptr.
        int a = 5; // a is now a variable and not a pointer anymore
        ptr = &a; } // Assign the address of a to the ptr.

int main() {
        int b = 7;
        int *ptr;
        *ptr = b;

        foo(ptr);

        std::cout << *ptr << std::endl;
        return 0;
}
```

c)

```
        int a = 4;
    int b = a + 6;
        int *p = &a, **q = &p;
        *p *= 4**&b***q;  // Never write such minified lines in your code!
        p = &b;
        *p = a + 5;
        (*p)++;
        p *= 2; // <-- error. Can't multiply an address
        cout << a << " " << b << " " << p << " " << (p + (p - 8)) << endl;
```

This code does not compile. p stores an address which can't be multiplied.
Also in the printout, two addresses can't be substracted or added together.
However if that was possible, the output would look like this:

```
Output: 640 646 0x16d7375e4 0x16d7375e4 + (0x16d7375e4 - 8)
```