1 Basic language constructs

The exercises here are (mostly) more theoretical in nature. Do note that some of these questions requires you to do some research on your own, since not everything is covered during the seminars.

1. Given

```
int n{};

int& foo(int& n)
{
  return n;
}

int bar(int& n)
{
  return n;
}

int&& move(int& n)
{
  return n;
}
```

   what is the value category of the following expression:

   (a) `n`

   (b) `n + 1`

   (c) `n = 2`

   (d) `double{}`

   (e) `move(n)`

   (f) `foo(n)`

   (g) `bar(n)`

   (h) `foo(n = bar(n))`

2. Write declarations for the following variables:

   (a) a pointer to `char`

   (b) an array with 10 `int`

   (c) a pointer to an array with 3 elements of type `std::string`

   (d) a pointer to a pointer to `char`

   (e) a constant `int`

(f) a pointer to a constant `int`

(g) a constant pointer to an `int`

3. Write a program that initializes all the variables from exercise 2.

4. For each of the following declarations: is it legal? If it is, what does it declare? The last three can be skipped since they are very complex.

   (a) `int a(int i);`

   (b) `int a(int);`

   (c) `int a(int (i));`

   (d) `int a(int (*i)());`

   (e) `int a(int* const);`

   (f) `int a const();`

   (g) `int a(int i());`

   (h) `int a(int const* (*)());`

   (i) `int a(int ());`

   (j) `int a(int (*)(int));`

   (k) `int a(int (*i)(int)[10]);`

   (l) `int a(int (*i[10])());`

   (m) `int a(int (&(*i)())[10]);`

   Note: Please, please, **PLEASE** don't ever do these kinds of declarations! Let this exercise serve as a demonstration as to what might happen if you don't think about readability. See next exercise for how you can make it more readable.

5. What does the following mean? How can it be used?

```
using x = int(&)(int, int);
```

6. Which size does the character array `msg` have? Which length does the string `"Hello world!"` have?

```
char msg[] { "Hello world!" };
```

7. Which of these are valid variable initializations? For those that are valid, what are their values? For those that are invalid explain why.

   - `int i1{};`
   - `int i2(2);`
   - `int i3 = 1;`
   - `int i4 = {};`

- `int i5();`
- `std::string str1{};`
- `std::string str2("hello");`
- `std::string str4(3, 'a');`
- `std::string str5 = str2;`
- `float f1{5.37e100};`
- `float f2 = 5.37e100;`
- `float f3{1738335806};`

8. Explain all type conversions that occur in this example.

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

int sum(double const* numbers,
        unsigned long long size)
{
  double result{};
  for (unsigned i{}; i < size; ++i)
  {
    result += static_cast<int>(numbers[i]);
  }
  return result;
}

int main()
{
  std::string message{};
  message = "Enter a number: ";

  double numbers[3];
  for (int i{0}; i < 3; ++i)
  {
    std::cout << message;
    if (!(std::cin >> numbers[i]))
    {
      return true;
    }
  }

  std::cout << sum(numbers, 3) + 1.0 << std::endl;
}
```

9. Explain all conversions in the following example. Why do they occur?

```cpp
#include <iostream>
int foo()
{
  return 0;
}

using function_t = int (*)();

int main()
{
  function_t f {main};

  char c{'A'};
  std::cout << c + 1 << std::endl;
  std::cout << c + 'A' << std::endl;

  std::cout << 0 - 1u << std::endl;

  f = foo;
  std::cout << f() << std::endl;
  std::cout << f << std::endl;
  std::cout << reinterpret_cast<void*>(f) << std::endl;
}
```