Exercise Class 7

1 Floating Point Number Systems

1.1 Exercise: Addition

1.1.1 Task

Consider the normalized floating point number system

$$F^* (\beta, p, e_{\min}, e_{\max})$$

with $\beta = 2$, $p = 3$, $e_{\min} = -4$, $e_{\max} = 4$.

Compute the following expressions as the parentheses suggest, representing each intermediate result (and the final result) in the normalized floating point system according to the rules of computing with floating point numbers.

$(10 + 0.5) + 0.5$

	dezimal		binary
	10		?????
+	0.5		?????
=			?????
+	0.5		?????
=	??	←	?????

$(0.5 + 0.5) + 10$

	dezimal		binary
	0.5		?????
+	0.5		?????
=			?????
+	10		?????
=	??	←	?????

1.1.2 Solution

$(10 + 0.5) + 0.5$

	dezimal		binär / binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	dezimal		binär / binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.0 \cdot 2^0$
$=$	12	$\leftarrow$	$1.10 \cdot 2^3$

2 References

References allow us to build an alias for an already existing object. They can be used as follows:

```
int a = 3;
int& b = a;
b = 2;
std::cout << a; // Output: 2
```

References are usually used as function parameters or return values. If the parameters of a function are not of reference type, we say that we "pass them to the function by value". This is what we did in all of our functions until now. For example:

```
int foo (int i) { ... }
```

In this case the function makes its own copies of the values, and uses these copies to do something in the function body. If the paramaters of a function are of reference type, hence will become aliases of the call arguments, we say that we "pass the arguments by references". For example:

```
int foo (int& i) { ... }
```

There at least three cases where it is useful to use references:

1. For implementing functions that need to return more than one result. For example:

```
// POST:  return value is the number of distinct real solutions of the quadratic
//        equation ax^2 + bx + c = 0. If there are infinitely many solutions
//        (a = b = c = 0), the return value is -1. Otherwise, the return value
//        is a number n from {0,1,2} and the solutions are written to s1, ... , sn
int solve_quadratic_equation (const double a, const double b, const double c,
                              double& s1, double& s2)
```

2. Passing a reference to a function we avoid copying the parameter. All a reference does, is to tell the program the location of the variable which the reference references. Now imagine you have a function that reads one entry in a gigantic vector:

```
void read_i (Vector& v, unsigned int i);
```

The vector `v` better be a reference, else one would copy the whole thing everytime an entry is read. If we use references it just tells the function where to find the one original vector so it can read the entry.

3. Sometimes it is impossible to copy something. `std::cout` is an example of this: There is only one output stream, we cannot make a copy of it like we could make a copy of an integer.

```
int a = 5;
int b = a;  // making a copy of an int
std::ostream o = std::cout; // Error: copying std::cout is impossible
std::ostream& o = std::cout; // This works, try it!
```

Not only function parameters can be references, but also references can be used as function return types. This is, for example, useful for:

```
int& increment (int& m) {
  return ++m;
}

int main () {
  int n = 3;
  increment (increment (n));
  std::cout << n << "\n"; // outputs 5
  return 0;
}
```

Here, the return type of the inner `increment (n)` is an `int` reference, returning `n` as an lvalue. The outer `increment` function on the other hand absorbs the same l-value as its argument, resulting in $n = 5$.