First attempt rated by:

Steffen Beudeker - s2503166
Jos Peters - s2701634

Resubmission Week 1

Exercise 3.
This exercise was labeled as “Won’t compile”, since the last word of ‘Raw String Literal’ appeared on a new line. We now lined out the text better. We compiled the code on both our macs and it worked fine (but only when we used –std=c++11 in the command line).

Remove the NTBs and transform them into RSLs

```
#include <iostream>

namespace {
    char const text[] =  // define the Raw String Literal
    R"(
    ^\\s+Encryption key:(\\w+)
    ^\\s+Quality=(\\d+)
    ^\\s+E?SSID:\"([[:print:]]+)\
    ^\\s+ssid=\"([[:print:]]+)\

    )";
}
int main()
{
    std::cout << text;
}
```

The rule for using the R”(…)” Format
The Raw String Literal format can only be used when using a recent version of C++ with your compiler. After some research, the RSL format requires at least C++11 to compile.

Exercise 5

Three of our statements contained a mistake. The first was not covered, the second was BABO and the third was not according to the exercise. We fixed the BABO problem, and eliminated the other two statements.

Is an unsigned value odd or even?

For this exercise we defined a main function, reading an unsigned integral value from cin. Then six cout statements were provided. The listing is provided below:

```
#include <iostream>
#include <math.h>

using namespace std;

int main() {
     int value;
     cin >> value;

     cout << ( value % 2 == 1 ? "odd" : "even") << '\n';
     cout << ( (value / 2 * 2) == value ? "even" : "odd")
     << '\n';
     cout << ( (value & 1) == 1 ? "odd" : "even") <<
     '\n';
     cout << ( (( value >> 1) << 1 ) == value ? "even" :
     "odd") << '\n';

}
```

With each statement provide a short sematic comment explaining why the expression correctly performs its task.

We will shortly describe the expressions in the chronological order displayed above. The first expression uses the modulo operator. If the remainder after division by 2 equals 0 the number is odd, otherwise it is even. The second line uses the divide operator. After division the value will be rounded off to a natural number. Since even numbers are dividable by 2, they do not need to be rounded off. Therefore only numbers that change after division and multiplication by 2 will be odd. The third expression uses a bitwise operator. Only the lowest-order bit determines if a number is odd or even, since all other bits are multiples of two. The operator & is used to determine if that bit was even or odd. The fourth line uses the same bitwise approach. This time the right shift operator makes the last digit disappear. When we shift it back with a left shift operation, we can see if the number changed. If that is the case, the number is odd since the ‘odd’ bit was used.

Exercise 7

For this exercise we learn to use bit-wise operators. Write a program that reads an unsigned value from the standard input stream.

```
#include <iostream>

using namespace std;

int main() {
     //Declare value
     int value;
     //Retrieve value from input stream
     cin >> value;
     //Print text to output stream
     cout << "the value x is "
     //Test for exact power of two and print to output
     stream
     << ((value & (value – 1)) == 0 ? "" : "not ")
     //Finish printing to output stream
     << "an exact power of two" << "\n";
}
```