Steffen Beudeker - s2503166
Jos Peters - s2701634

Exercise 1.
For this exercise we need to construct the classic 'Hello World' code, used as starting point in the first book of Kernighan and Ritchie.

The listing:

```
#include <iostream>

int main()
{
    std::cout << "Hello World\n";
}
```

The commands entered to compile and link the program:

```
g++ -Wall -c hello.cc
g++ -Wall hello.o -o hello
```

A short description of object file and executable:
The object file is called 'hello.o'. It is generated by the compiler from the source file and contains external references, and definitions. The file is not readable for humans, I tried it but got a headache after the second line. The executable program is called 'hello'. It is generated by the linker and contains the final program that gives 'Hello World' as output.

The program output:

```
(Command: ./hello >>file_1.txt)
Hello World
```

The size in bytes:

```
The source file (hello.cc)          = 69
The object file (hello.o)           = 1.700
The executable program (stripped)   = 8.792
The file iostream                   = 2.695
```

Exercise 2.
In this exercise, questions about core concepts will be answered.

What's the difference between a declaration and definition?
When defining a variable, memory space will be reserved for the variable. With the declaration of a variable the existence of the variable will only be announced.

What are header files used for?
A header file is a file that contains many external functions which can be included by your pre-compiler in your

program when using the #include directive in your source code. Header files usually have the extension ".h" and by convention usually only contain declarations and no definitions.

In what part of the construction process does a compiler use header files and in what part libraries? (why does the compiler use header files and libraries)

The header files are read by the compiler immediately from the source file. Thereafter, the linker will use the libraries if they are needed by the object file. (The header files make life easier by declaring functions, the libraries take more time and memory space and follow up in the later stage)

Is a library an object module?

While a library can contain object modules, it also contains other resources such as configuration data, documentation and more. This is the reason that a library cannot be called an object module.

Why is an object module obtained after compiling a source containing int main() not an executable program?

An object module is not executable because the linker still has to piece together all the objects. When a program in C tries to use a function that is contained in one of the libraries, the linker has to search for this function to be able to successfully complete the program.

Exercise 3

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 4

What are 'escape sequences'?

Escape sequences start with a \. They can be used to conduct special actions, such as starting a newline. The escape sequence is a character that is translated into another character, hence the escape sequence doesn't represent itself.

What standard 'escape sequences' are defined in C++? What are their meanings?

Sequence:	Meaning
\a	alert
\b	backspace
\f	form feed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
? or \?	question mark
\'	single quote
\"	double quote
\0	the null character
\ooo	octal
\xhhh	hexadecimal
\uxxxx	Unicode (UTF-8)
\Uxxxxxxxx	Unicode (UTF-16)
\nnn	arbitrary octal value
\e	escape character

What happens if another character is written as an escape sequence?

Other characters that follow after a \ will literally be represented without the \. This is why the single quote in the table above is represented by \', and the backslash as \ \.

Exercise 5

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 toEvenOrNotToEven(int value)
{
  if(value == 0)
      return 1;
  else if(value == 1)
      return 0;
  else if(value<0)
      return toEvenOrNotToEven(-value);
  else
      return toEvenOrNotToEven(value-2);
}

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';
  cout << ( toEvenOrNotToEven(value) == 0 ? "odd" : "even")
  << '\n';
  cout << ( pow(-1,value) == -1 ? "odd" : "even") << '\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 used. 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. The fifth line uses the function that was written at the top of the code. It uses recursion to determine if the number is odd or even. It subtracts 2 from the number until it is 0 or 1. If it is 0 the number is odd, and otherwise it is even. Finally, the last line uses a classic mathematical trick. If -1 to the power of the number is negative, the number is odd and otherwise it is even.