Beginselen van Programmeren
Exercise Session 3:
iterations

1 Warm up exercises

E1. Write a program that prints the sum of the first n natural numbers: $1 + 2 + \ldots + n$. In fact, write two versions, one that uses a while loop, and one that uses a for loop. Your output should look like this:

Enter the number of terms: 10
The sum of the first 10 natural numbers is 55.

How are you going to test your program to check if it is correct?

E2. Write a program that converts temperatures in Celsius to Fahrenheit. The program should accept a stream of temperatures from the user and stop when the user enters the letter q.
Can you use a for loop for this exercise? Why (not)?

2 Now let's continue...

E3. Write a program that reads a sequence of integers (stops accepting input when the user enters 0) and computes the alternating sum of all elements. For example, if the program is executed with the input data 1 4 9 16 9 7 4 9 11 0, then it computes $1-4+9-16+9-7+4-9+11 = -2$.

E4a. A prime number is one that is not divisible by any number other than 1 and itself. Write a program that asks the user to input a positive integer and checks whether the given integer is a prime number.
Use a while loop in your solution.
How are you going to test your program to check if it is correct?
E4b. Extend the above program to input an integer and print all prime numbers up to that integer.
Use a for loop in your solution.

E5. Write a program to input a number and reverse the order of digits. For example if the input to the program is 12345, the output should be 54321. (*Do not use string*)

E6a. Write a program that computes the factorial of a positive integer. $n! = 1 * 2 * 3 * \cdots * n$
E6b. Use the above program to calculate e^x. The value of e^x can be computed as the power

series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{1}$$

Of course, you can't compute an infinite sum. Just keep adding values until an individual summand (term) is less than a certain threshold.

E7. Credit Card Number Check: The last digit of a credit card number is the check digit, which protects against transcription errors such as an error in a single digit or switching two digits. The following method is used to verify actual credit card numbers but, for simplicity, we will describe it for numbers with 8 digits instead of 16:

1. Take the sum of alternate digits of the credit card number, beginning with the rightmost digit. For example, if the credit card number is 4358 9795, then you form the sum $5 + 7 + 8 + 3 = 23$.

2. Double each of the digits that were not included in the preceding step. Add all digits of the resulting numbers. For example, with the number given above, doubling the digits, starting with the next-to-last one, yields 18 18 10 8. Adding all digits in these values yields $1 + 8 + 1 + 8 + 1 + 0 + 8 = 27$.

3. Add the sums of the two preceding steps. If the last digit of the result is 0, the number is valid. In our case, $23 + 27 = 50$, so the number is valid.

Write a program that implements this algorithm. The user should supply an 8-digit number, and you should print out whether the number is valid or not. If it is not valid, you should print out the value of the check digit that would make the number valid.

Here are sample program runs:

Enter 8 digit credit card number: 43589794
The credit card number is not valid.
The last digit should be 5

Enter 8 digit credit card number: 43589795
The credit card number is valid.

E8. Write a program to display the multiplication table. The output should look like:

```
    1   2   3   4   5   6   7   8   9   10  11  12
1   1   2   3   4   5   6   7   8   9   10  11  12
2   2   4   6   8   10  12  14  16  18  20  22  24
3   3   6   9   12  15  18  21  24  27  30  33  36
4   4   8   12  16  20  24  28  32  36  40  44  48
5   5   10  15  20  25  30  35  40  45  50  55  60
6   6   12  18  24  30  36  42  48  54  60  66  72
7   7   14  21  28  35  42  49  56  63  70  77  84
8   8   16  24  32  40  48  56  64  72  80  88  96
9   9   18  27  36  45  54  63  72  81  90  99  108
10  10  20  30  40  50  60  70  80  90  100 110 120
11  11  22  33  44  55  66  77  88  99  110 121 132
12  12  24  36  48  60  72  84  96  108 120 132 144

Process finished with exit code 0
```

E9. Write a program which asks the user for a height of a diamond of stars and then draws one. The program executes continuously until the user enters a value smaller than one. A sample run might look like:

```
Enter the height of the diamond: 7
   *
  ***
 *****
*******
 *****
  ***
   *

Enter the height of the diamond: 6
   *
  ***
 *****
 *****
  ***
   *

Enter the height of the diamond: 2
 *
 *

Enter the height of the diamond: 0

Process finished with exit code 0
```

E10. Write a program that sorts a list of integers in ascending order. The program should accept a stream of temperatures from the user and stop when the user enters the letter q.
It is not allowed to use built in sorting functions, make use of loops and your favourite sorting algorithm instead. (e.g. selection sort, bubble sort, insertion sort, quicksort, merge sort)

E11. Write a program that takes a list of integers as input. Output a list of the same size where every element is replaced by the highest value of either itself of one of its direct neighbours. An example is given below.
input: 34 12 37 77 6 22 15 3 8 5
output: 34 37 77 77 77 22 22 15 8 8

E12. Write a program that prints the coordinates of all saddle points in a given matrix. A saddle point is defined by having strictly bigger neighbours in one direction (horizontal or vertical) and having strictly smaller neighbours in the other direction. In the example matrix below the saddle points are denoted in bold. Choose a convenient way of inputting matrices into your program and outputting the results.

5	8	46	2	99
15	4	**23**	7	84
5	4	35	96	3
25	**6**	45	1	3
3	5	8	4	6