SD11 Q1 Assignments – Week 5-6

Version: September 2018

Table of Contents

Assignments must be finished before the week 7 practicum!

WEEK 5-6

Assignment 5-6.1: Tabular output

Consider a program that asks the user for a positive integer value. The program should then print a table like the one shown below (in this example, the user has entered value 20).

a) What kind of repetition will you use and why?
b) Provide the flow chart of the program.
c) Write the program.

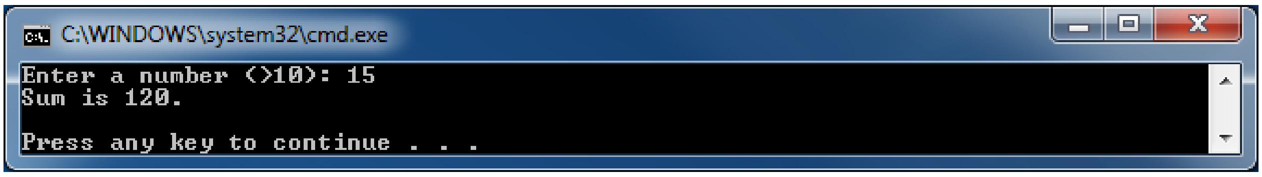
Hint: use the tab escape sequence `\t` to format the output. For example, the string `"N\tN*N\tN*N*N\tN*N*N*N"` will display the header of the table shown below.

```
N       N*N     N*N*N     N*N*N*N

1       1       1         1
2       4       8         16
3       9       27        81
…
20      400     8000      160000
```

Assignment 5-6.2: Sum of integers

Consider a program that asks the user for an integer value greater than 10, say 15. The program should then calculate the sum of all positive values up to 15: 1 + 2 + 3 + … + 15.

A possible result of your program could be something like:


C:\WINDOWS\system32\cmd.exe

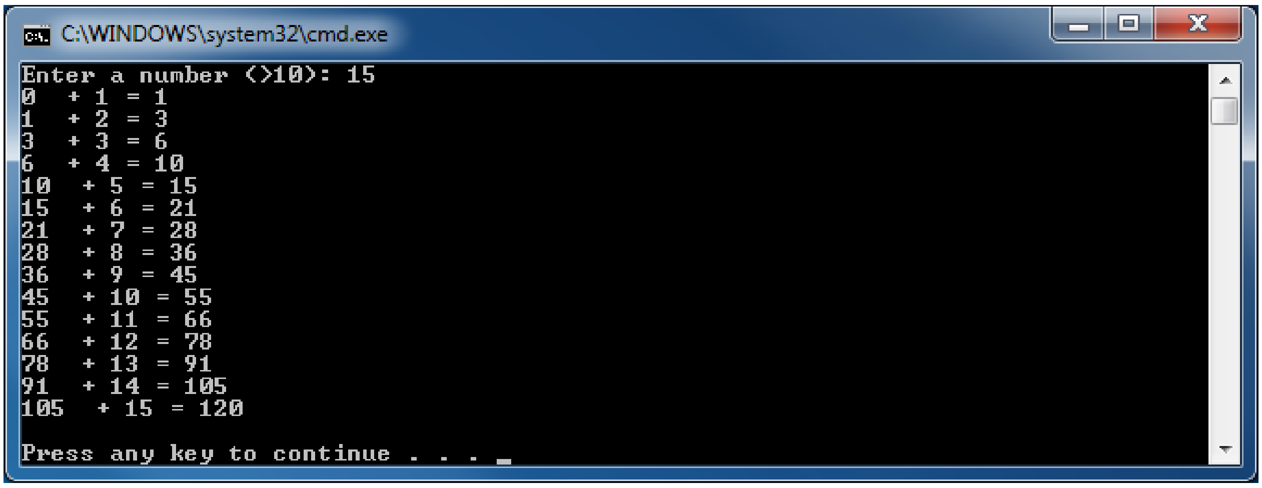
```
Enter a number (>10): 15
Sum is 120.

Press any key to continue . . .
```

a) What kind of repetition will you use and why?
b) Provide the flow chart of the program.
c) Write the program.

Assignment 5-6.3: Advanced Sum of integers (extension of assignment 5-6.2)

Adjust the flow chart and the program you wrote for assignment 5-6.2 in order to show the current sum after adding each number.

A possible result of your program could be something like:



```
Enter a number (>10): 15
0  + 1 = 1
1  + 2 = 3
3  + 3 = 6
6  + 4 = 10
10  + 5 = 15
15  + 6 = 21
21  + 7 = 28
28  + 8 = 36
36  + 9 = 45
45  + 10 = 55
55  + 11 = 66
66  + 12 = 78
78  + 13 = 91
91  + 14 = 105
105  + 15 = 120

Press any key to continue . . . _
```

Assignment 5-6.4: Reverse sum of integers

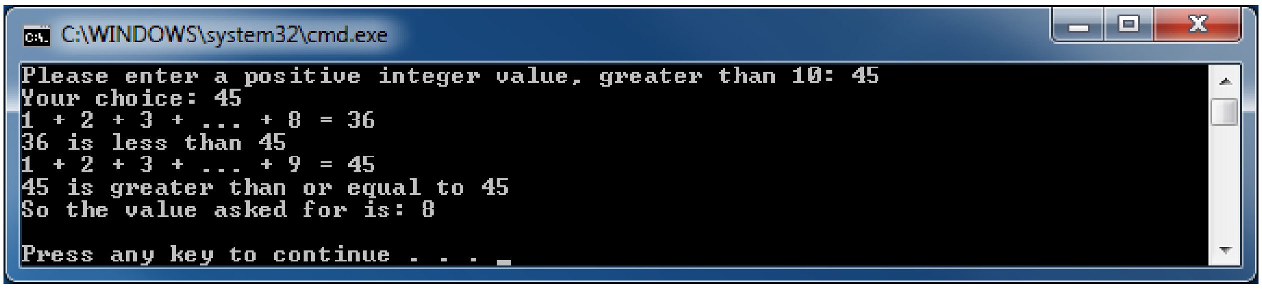
Consider a program that asks the user for an integer value greater than 15, say 45. The program should then calculate the **`int`** value n for which it holds that:

(1). $1 + 2 + 3 + \ldots + (n - 1) + n < 45$

and

(2). $1 + 2 + 3 + \ldots + (n - 1) + n + (n + 1) >= 45$

A possible result of your program could be something like:



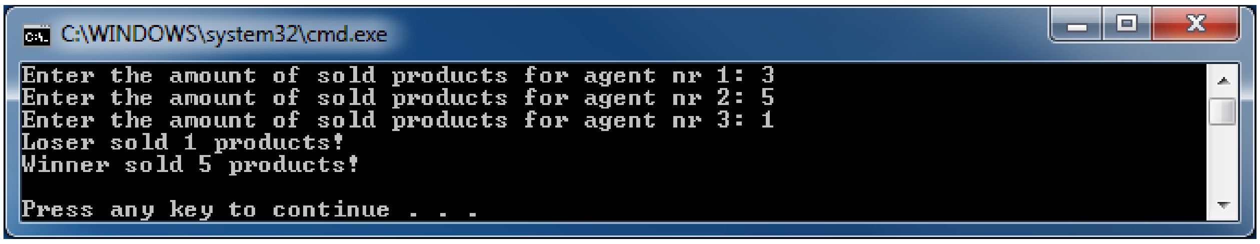
```
Please enter a positive integer value, greater than 10: 45
Your choice: 45
1 + 2 + 3 + ... + 8 = 36
36 is less than 45
1 + 2 + 3 + ... + 9 = 45
45 is greater than or equal to 45
So the value asked for is: 8

Press any key to continue . . . _
```

a) What kind of repetition will you use and why?
b) Provide the flow chart of the program.
c) Write the program.

Assignment 5-6.5: Limited minimum and maximum (based on exercise 5.21 from the book)

The algorithm for finding the smallest (minimum) and largest (maximum) number is often used in programming. For example, a program that determines the winner and loser of a sales competition would input the number of products sold by each sales person. The person who sold the largest number of products wins the competition. The person who sold the smallest number of products loses the competition.

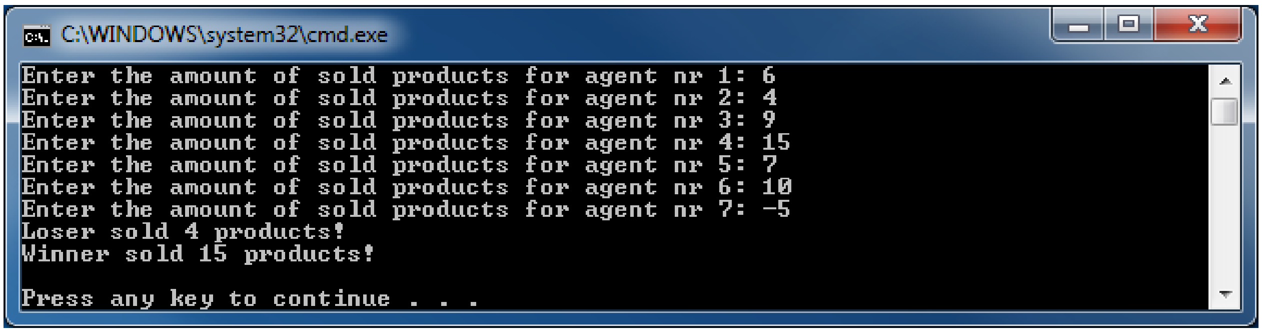
The program should ask the user to input the amount of sold products of 3 sales agents. Next, the program should show which amount is the largest and which is the smallest. A possible result of your program could be something like:



```
Enter the amount of sold products for agent nr 1: 3
Enter the amount of sold products for agent nr 2: 5
Enter the amount of sold products for agent nr 3: 1
Loser sold 1 products!
Winner sold 5 products!

Press any key to continue . . .
```

a) Provide the flow chart of this program.
b) Implement this program.

Adjust the program you made in such a way that the user can enter as many sales agents as he/she wants. The user should indicate that all sales agents have been processed by entering a negative number. A possible result of your program could be something like:



```
Enter the amount of sold products for agent nr 1: 6
Enter the amount of sold products for agent nr 2: 4
Enter the amount of sold products for agent nr 3: 9
Enter the amount of sold products for agent nr 4: 15
Enter the amount of sold products for agent nr 5: 7
Enter the amount of sold products for agent nr 6: 10
Enter the amount of sold products for agent nr 7: -5
Loser sold 4 products!
Winner sold 15 products!

Press any key to continue . . .
```

c) What kind of repetition structure would you use? Explain why?
d) Provide the flow chart of this extended program.
e) Implement the code.

Optional: extend the program even further, such that not only the number of products sold is shown for the winner and loser, but also the number of the sales agent who is the winner and the number of the sales agent who is the loser. So for example:

Sales agent 2 is the loser and sold 4 products!
Sales agent 4 is the winner and sold 15 products!

If the same number of sold products is entered for two sales agents, you can choose which one is the winner (you don't need to take special care for this in your program).

Assignment 5-6.6: Number π

The German mathematician Leibniz (1646–1716) discovered the rather remarkable fact that the mathematical constant π can be computed using the following mathematical relationship:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$$

The formula to the right of the equal sign represents an infinite series; each fraction represents a term in that series. If you start with 1, subtract one-third, add one-fifth, and so on, for each of the odd integers, you get a number that gets closer and closer to the value of π/4 as you go along.

Design and write a program that calculates an approximation of π consisting of the first 10,000 terms in Leibniz' series.

Assignment 5-6.7: Predator – prey

In a predator-prey simulation, you compute the populations of predators and prey, using the following equations:

$$prey_{n+1} = prey_n \times (1 + A - B \times pred_n)$$
$$pred_{n+1} = pred_n \times (1 - C + D \times prey_n)$$

Here, *A* is the rate at which prey birth exceeds natural death, *B* is the rate of predation, *C* is the rate at which predator deaths exceed births without food, and *D* represents predator increase in the presence of food.

We will write a program that prompts users for these rates, the initial population sizes, and the number of periods the simulation should run. Then it should print the populations for the given number of periods.

As inputs, try *A* = 0.1, *B* = *C* = 0.01, and *D* = 0.00002 with initial prey and predator populations of 1,000 and 20.

a) What kind of repetition structure would you use? Explain why?
b) Provide the flow chart of this program.
c) Implement the code.

Assignment 5-6.8: Number guessing game

We are going to make a game where the user has to guess a (random) number greater than or equal to 0 and less than 100. As we did not yet discuss random number generation, you may have your "random" number hardcoded in the declaration of the variable (use `const int`).

Your program should now ask the user to guess the number. If the user guesses the number correctly, the program should show "Correct!". Otherwise, the program should check whether the guessed number is lower or higher than the random number:

- If the guessed number is less than the random number, the program outputs "Higher!"
- If the guessed numer is greater than the random number, the program outputs "Lower!"

After that, the program should ask the user for his/her next guess. This should continue until the user guesses the correct number.

a) What kind of repetition structure would you use? Explain why!
b) Provide the flow chart of this program.
   Note: use plain English in the blocks of your flowchart, not C++ code.
c) Implement the code.

Now we are going to extend our program in two steps:

Extension 1: In the program you implemented, the user is given as many tries as needed to guess the correct number. Redesign the program so that the user has no more than five tries to guess the correct number. At the end, your program should display an appropriate message, such as "You win!" or "You lose!"

Extension 2: In the current program, if the guessed number is not correct, the program only tells you to guess lower or higher. We are now going to modify our program to give you a bit more help.

Introduce a new variable **`int`** `diff`, and after each guess calculate the difference between the user's guess and your random number. Suppose your random number is stored in a variable named num and the user's guess is stored in a variable named guess, the absolute value of the difference can be calculated as:

```
diff = abs(num – guess)
```

In order for this to work, you need to include the `cstdlib` library in your code.

Now, if `diff` is equal to 0, the user guessed the number correctly and wins the game (as before). If `diff` is not equal to 0, the program outputs a message as follows:

- If `diff` ≥ 50, the program displays "Much, much higher!" or "Much, much lower!"
- If 30 ≤ `diff` < 50, the program displays "Much higher!" or "Much lower!"
- If 10 ≤ `diff` < 30, the program displays "Higher!" or "Lower!"
- If 0 < `diff` < 10, the program displays "Slightly higher!" or "Slightly lower!"

As in the previous version of your program, the user should get five tries to guess the number.

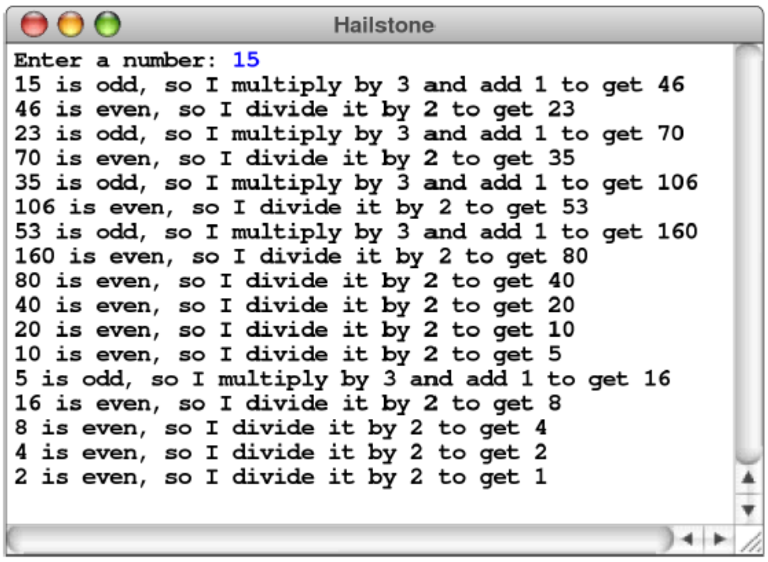
Assignment 5-6.9: Hailstone sequence (optional)

In 1979, Douglas Hofstadter, Professor of Cognitive Science at the University of Indiana, wrote Gödel, Escher, Bach, which he described as “a metaphorical fugue on minds and machines in the spirit of Lewis Carroll.” The book won the Pulitzer Prize for Literature and has over the years become one of the classics of computer science. Much of its charm comes from the mathematical oddities and puzzles it contains, many of which can be expressed in the form of computer programs. Of these, one of the most interesting concerns the sequence of numbers formed by repeatedly executing the following rules for some positive integer *n*:

- If *n* is equal to 1, you’ve reached the end of the sequence and can stop.
- If *n* is even, divide it by two.
- If *n* is odd, multiply it by three and add one.

Although it also goes by several other names, this sequence is often called the **hailstone sequence** because the values tend to go up and down before coming back to 1, much as hailstones do in the clouds in which they form.

Design and write a program that reads in a number from the user and then generates the hailstone sequence from that point, as in the following sample run:

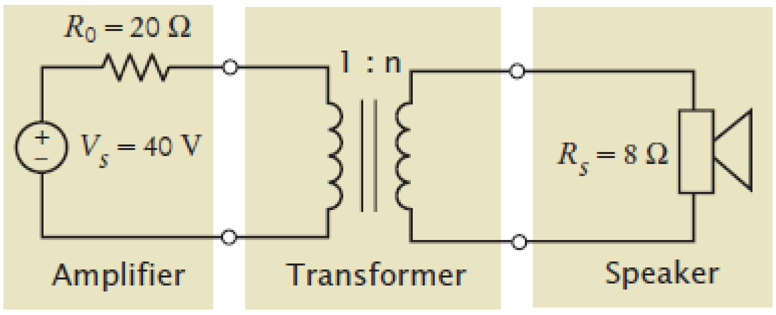


```
Enter a number: 15
15 is odd, so I multiply by 3 and add 1 to get 46
46 is even, so I divide it by 2 to get 23
23 is odd, so I multiply by 3 and add 1 to get 70
70 is even, so I divide it by 2 to get 35
35 is odd, so I multiply by 3 and add 1 to get 106
106 is even, so I divide it by 2 to get 53
53 is odd, so I multiply by 3 and add 1 to get 160
160 is even, so I divide it by 2 to get 80
80 is even, so I divide it by 2 to get 40
40 is even, so I divide it by 2 to get 20
20 is even, so I divide it by 2 to get 10
10 is even, so I divide it by 2 to get 5
5 is odd, so I multiply by 3 and add 1 to get 16
16 is even, so I divide it by 2 to get 8
8 is even, so I divide it by 2 to get 4
4 is even, so I divide it by 2 to get 2
2 is even, so I divide it by 2 to get 1
```

One of the fascinating things about the hailstone sequence is that no one has yet been able to prove that the process always stops. The number of steps in the process can get very large, but somehow, it always seems to climb back down to one.

Assignment 5-6.10: Transformer (optional)

The photo at left shows an electric device called a “transformer”. Transformers are often constructed by wrapping coils of wire around a ferrite core. The figure below illustrates a situation that occurs in various audio devices such as cell phones and music players. In this circuit, a transformer is used to connect a speaker to the output of an audio amplifier.



The symbol used to represent the transformer is intended to suggest two coils of wire. The parameter *n* of the transformer is called the “turns ratio” of the transformer. (The number of times that a wire is wrapped around the core to form a coil is called the number of turns in the coil. The turns ratio is literally the ratio of the number of turns in the two coils of wire.)

When designing the circuit, we are concerned primarily with the value of the power delivered to the speakers—that power causes the speakers to produce the sounds we want to hear. Suppose we were to connect the speakers directly to the amplifier without using the transformer. Some fraction of the power available from the amplifier would get to the speakers. The rest of the available power would be lost in the amplifier itself. The transformer is added to the circuit to increase the fraction of the amplifier power that is delivered to the speakers.

The power, P_s, delivered to the speakers is calculated using the formula:

$$P_s = R_s\left(\frac{nV_s}{n^2R_0 + R_s}\right)^2$$

The C++ program that you will create, models the circuit shown and varies the turns ratio from 0.01 to 2 in 0.01 increments, then determines the value of the turns ratio that maximizes the power delivered to the speaker.

a) What kind of repetition structure would you use? Explain why?
b) Provide the flow chart of this program.
c) Implement the code.