L04 Tasks

L04-T1: Different repetition structures

L04-T2: Repetition structure with string

L04-T3: Repetition structure with multiple termination conditions

L04-T4: Extensions of repeat structures

L04-T5: Menu-based program, continuing from exercise L03-T3

Submission:

- Submit L04-T4 and L04-T5 to CodeGrade via Moodle before the deadline.

Note:

- Be especially careful with spaces so that your output follows the sample output. Note that also each input-string in the program is ending with '\n'. The reason for this is that it makes the output in CodeGrade more readable.

- If you see some "Code Structure Tests" hidden in CodeGrade but they are not mentioned in the task description, you don't need to worry about them. They are just there to make sure the code is ok.

L04-T1: Different repetition structures

Write a program that asks the user for two integers: a start value and an end value. After that, the program prints all the values between these numbers according to the example run below, including the initial and final values. Implement the repetition first with a “for”-loop and then with a “while”-loop, i.e., test both repetition structures and compare their implementation. In the loop, you should always print the numbers so that you have all the numbers neatly on the same line (see examples below).

Example run 1:

```
Enter the staring value:
1
Enter the ending value:
10
Implementation with a for loop:
1 2 3 4 5 6 7 8 9 10
Implementation with a while loop:
1 2 3 4 5 6 7 8 9 10
```

L04-T2: Repetition structure with string

Commonly it is agreed that there are five vowels in the English language: a, e, i, o, u. Write a program to count the number of vowels in the given string. Note that both lower and upper case letters are considered vowels.

Example run 1:

```
Enter a string:
The USA, the shorthand for the United States of America.
Number of vowels is: 18
```

L04-T3: Repetition structure with multiple termination conditions

Write a program that prints numbers based on specific rules. Initially, you'll ask the user to input two numbers, which we'll call "a" and "b". After that, you'll start printing these numbers in a loop. In each round of the loop, the value of "a" will double, and the value of "b" will increase by 100. The loop will stop when either "a" or "b" (or both) becomes greater than 1000. Your program should also tell the reason that caused the loop to end.

Example run 1:

```
Enter a:
5
Enter b:
333
a: 5 b: 333
a: 10 b: 433
a: 20 b: 533
a: 40 b: 633
a: 80 b: 733
a: 160 b: 833
a: 320 b: 933
b exceeded 1000
```

Example run 2:

```
Enter a:
2
Enter b:
100
a: 2 b: 100
a: 4 b: 200
a: 8 b: 300
a: 16 b: 400
a: 32 b: 500
a: 64 b: 600
a: 128 b: 700
a: 256 b: 800
```

```
a: 512 b: 900
a exceeded 1000
```

Example run 3:

```
Enter a:
2
Enter b:
111
a: 2 b: 111
a: 4 b: 211
a: 8 b: 311
a: 16 b: 411
a: 32 b: 511
a: 64 b: 611
a: 128 b: 711
a: 256 b: 811
a: 512 b: 911
a exceeded 1000
b exceeded 1000
```

L04-T4: Extensions of repeat structures

(Submit this task to CodeGrade on Moodle)

In this task we will practice for, break, continue, and if structures.

(Note: it is mandatory to have for-loop structure, `break` and `continue` in the code - this will be tested in the "Code structure test" in CodeGrade. If the "Code structure test" fails, the remaining tests will not be graded.)

Write a program that searches the user-defined integer range for an integer that is divisible by five and seven. In the beginning, the user enters the lower and upper limits of the search range into the program, after which the program starts testing numbers one at a time. The program should print the conclusions for each number: For example, "20 is NOT divisible by seven, rejecting.", "21 is NOT divisible by five, rejecting.", ..., "The number 35 is divisible by 5 and 7".

The program should be implemented in such a way that **if the number is not divisible by five, we move directly to testing the next number**. The search is stopped as soon as the program finds a suitable number. If a suitable number is not found in the entire search area, the message "No suitable value was found in the range" is printed.

This can be done, for example, with a helper variable that is initialized at the beginning of the program with the value False and changed to the value True if the number is found. When the search is finished, the program prints the above notification if necessary. See the exact format of the output in the example run below.

Example run 1:

```
Enter the lower limit of the range:
99
Enter the upper limit of the range:
113
99 is NOT divisible by 5, rejecting.
100 is NOT divisible by 7, rejecting.
101 is NOT divisible by 5, rejecting.
102 is NOT divisible by 5, rejecting.
103 is NOT divisible by 5, rejecting.
104 is NOT divisible by 5, rejecting.
The number 105 is divisible by 5 and 7.
Stopping the search.
```

Example run 2:

```
Enter the lower limit of the range:
107
Enter the upper limit of the range:
119
107 is NOT divisible by 5, rejecting.
108 is NOT divisible by 5, rejecting.
109 is NOT divisible by 5, rejecting.
110 is NOT divisible by 7, rejecting.
111 is NOT divisible by 5, rejecting.
112 is NOT divisible by 5, rejecting.
113 is NOT divisible by 5, rejecting.
114 is NOT divisible by 5, rejecting.
115 is NOT divisible by 7, rejecting.
116 is NOT divisible by 5, rejecting.
117 is NOT divisible by 5, rejecting.
118 is NOT divisible by 5, rejecting.
119 is NOT divisible by 5, rejecting.
No suitable value was found in the range.
```

L04-T5: Menu-based program, continuing from exercise L03-T3

(Submit this task to CodeGrade on Moodle)

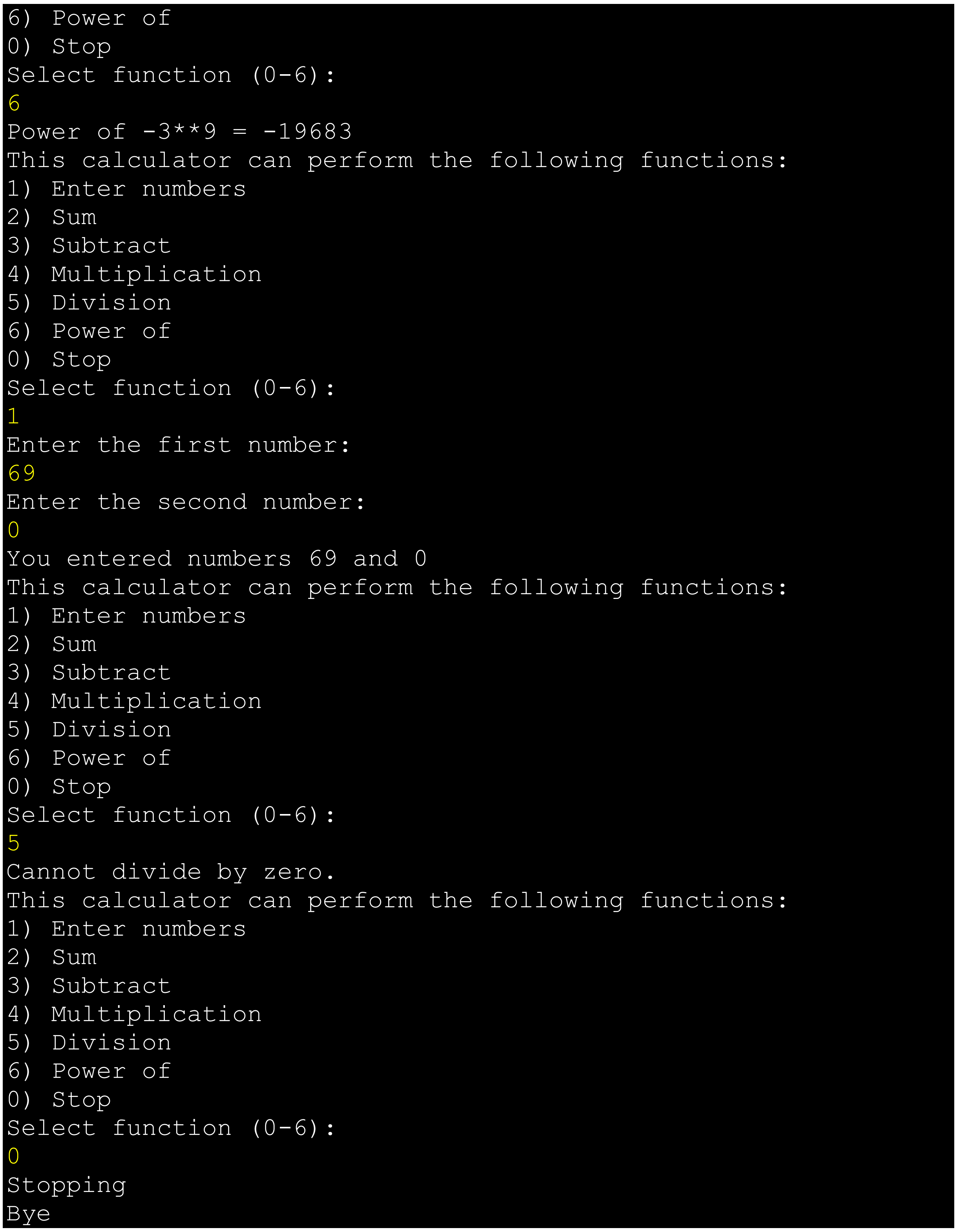
Extend the calculator of the task L03-T3 so that it can calculate several calculations until the user indicates that he wants to stop the execution of the program. At the same time, make the program more flexible from the user's point of view by putting the asking of the numbers as one item in the menu according to the example run below.

This means adding a repetition structure to an existing menu-based program to make the program easier to use, and at the same time modify the program to be more user-friendly.

When calculating division, round the result to two decimal precisions with the round function. When the user chooses to stop, the program execution stops, and the program says "Bye". If the user selects an unknown number, the program reports a problem by stating "Unknown selection, try again." and asks for a new selection.

Example run 1:

```
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
6) Power of
0) Stop
Select function (0-6):
7
Unknown selection, try again.
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
6) Power of
0) Stop
Select function (0-6):
1
Enter the first number:
-3
Enter the second number:
9
You entered numbers -3 and 9
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
6) Power of
0) Stop
Select function (0-6):
3
Subtraction -3 - 9 = -12
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
```



```
6) Power of
0) Stop
Select function (0-6):
6
Power of -3**9 = -19683
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
6) Power of
0) Stop
Select function (0-6):
1
Enter the first number:
69
Enter the second number:
0
You entered numbers 69 and 0
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
6) Power of
0) Stop
Select function (0-6):
5
Cannot divide by zero.
This calculator can perform the following functions:
1) Enter numbers
2) Sum
3) Subtract
4) Multiplication
5) Division
6) Power of
0) Stop
Select function (0-6):
0
Stopping
Bye
```