Week 3

Exercise 1: Function with parameters and return value

Write a program that calculates the exponent functions for integers. Ask the user for the ***base*** (integer) and the ***exponent*** (positive integer) in the main program. Pass them to the function as parameters, perform the exponent calculation using loop structure, and return the result of the calculation from the function to the main program as a return value. Print the result in the main program according to the example below.

Your program should be "***repetitive***," that is, it keeps on running until it is stopped by the user.

Add a new line to the end of each output line.

Example use case:

```
Enter the base (integer):
5
Enter the exponent (integer):
4
5^4 = 625.
Do you want to continue? (yes = y, no = any other key)
y
Enter the base (integer):
8
Enter the exponent (integer):
3
8^3 = 512.
Do you want to continue? (yes = y, no = any other key)
n
```

Exercise 2: handling files with functions

Write a program with three functions in addition to the main program. One function displays the menu, one function is for writing a file and the third one is for reading the file.

The program asks for the name of the file to be processed at the beginning (extension `.txt`). The name is passed to the writing and reading functions as a parameter.

The function that writes the file asks the user for first names of people and appends them to the end of the file. Check that there are no empty lines in your output file.

The function that reads the file, opens the file, reads the names, and prints them line by line.

The maximum length of the names you are asked to use is 48 characters.

If one tries to print the file before nothing is added there, this should raise an error. This can handled by printing the message "`Failed to open file, terminating`" and exiting the program by using exit(0) (see lecture slides). Note that `perror` is not working correctly in CodeGrade, so do not try to use that.

Add a new line to the end of each output line.

Example use case 1:

```
Enter the name of the file to be processed:
names.txt
Select from the options below
1) Add a new name
2) Print names
0) Exit
Enter your selection:
1
Enter a name to add:
Linus
Select from the options below
1) Add a new name
2) Print names
0) Exit
Enter your selection:
1
Enter a name to add:
Juhani
Select from the options below
1) Add a new name
2) Print names
0) Exit
Enter your selection:
1
Enter a name to add:
Teuvo
Select from the options below
1) Add a new name
2) Print names
0) Exit
Enter your selection:
2
Linus
Juhani
Teuvo
File read and printed.
Select from the options below
1) Add a new name
2) Print names
0) Exit
Enter your selection:
0
```

Example use case 2:

```
Enter the name of the file to be processed:
noFile.txt
Select from the options below
1) Add a new name
2) Print names
0) Exit
Enter your selection:
2
Failed to open file, terminating
```

Exercise 3: Comparing variables

Write a program that can compare integers, floats, and strings (write a function for each one) using a ***repetitive*** menu. Format used for printing floats is “%5.2f”. Use the **`strcmp`** function for comparing strings. Use `man` to find out how it works.

Example use case:

```
Select from the options below
1) Compare Integers
2) Compare Decimals
3) Compare Strings
0) Exit
Enter your selection:
1
Enter two integers:
34
35
Number 35 is bigger and 34 is smaller.
Select from the options below
1) Compare Integers
2) Compare Decimals
3) Compare Strings
0) Exit
Enter your selection:
2
Enter two decimal numbers:
345.13
231.123
Number 345.13 is bigger and 231.12 is smaller.
Select from the options below
1) Compare Integers
2) Compare Decimals
3) Compare Strings
0) Exit
Enter your selection:
3
Enter two strings:
Ville Kalle
```

```
As a string, 'Ville' is bigger and 'Kalle' is smaller.
Select from the options below
1) Compare Integers
2) Compare Decimals
3) Compare Strings
0) Exit
Enter your selection:
0
```

Exercise 4: Dealing with strings

We do our own version of the string type in this exercise. The program asks to input a string. We will use '#' as our own ending character, that is, each string inputted **must** end with '#'. To read the input string, use the `fgets()` function to get the entire line with spaces.

Write a function `strLenght(char s[])` that computes the number of characters in string `s`, but does not count '#'.

Write a function `copyString(char s1[], char s2[])` that copies the string `s1` to `s2`. When copying, use for-loop and `strLenght`. Be sure that our ending character `'#'` is also copied.

Write a function `printString(char s[])` that prints the string `s`. The function prints the string `s` character-by-character using `for`-loop and `strLenght`. The end character `'#'` is not printed. The function prints a new line after it has printed `s`.

In this exercise, using the `strcpy` and `strlen` functions and the `string.h` library are not allowed!

Example use case 1:

```
Enter a string to copy:
this is a string#
this is a string
this is a string
```

Example use case 2:

```
Enter a string to copy:
this is# as string
this is
this is
```

Exercise 5: Writing random numbers to a binary file

Write a program that writes and reads a file in ***binary*** format. The program must first generate 20 integers, save them in binary format to a file. If the storing was successful, then the program reads the saved file and prints the numbers to the screen.

At the beginning of the main program, ask the user for the name of the binary file to be processed (extension `.bin`).

Random numbers can be generated with the function **`int rand()`**. This algorithm needs to be initiated by giving a `seed`. This seed is given by the command `srand(seed)`. Your program should ask for this `seed` from the user after it has asked the file name. Note that the random number generator should only be seeded once.

If the file is written successfully, the main program calls the function that opens the written file, reads, and prints the stored numbers.

Add a new line to the end of each output line.

Example use case:

```
Enter the name of a binary file (extension .bin):
my_file.bin
Enter the seed for random number generator:
10
The file contains the following integers:
1215069295 1311962008 1086128678 385788725 1753820418 394002377
1255532675 906573271 54404747 679162307 131589623 179656373 2097642528
2100364173 1876662344 113981581 577648320 198294838 1891854071 1208116668
```