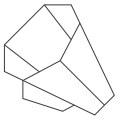


# Read this first!!

When doing the exercises, it is advised that you make notes! These notes will be useful for the assignment you have to write later on in the course.

# Exercise 3.1 Working with integers, floats and doubles

This exercise should illustrate what problems might occur if integers and operators such as division (/) and modulo (%) are used "incorrectly".

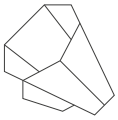
1. Implement a small program where you divide 7 with 3 and print the result. Also, find the remainder using the modulo operator. First, use int's as types variables to hold the values. Then try to fix the problems observed using other datatypes.
2. Document your findings.

# Exercise 3.2 How does string (character arrays) work in C?!

This exercise is meant to illustrate how the basics of strings (char-arrays) work in C.

```
#include <stdio.h>
int main () {
  char text[] = "The quick brown fox jumps over lazy dog";
  char* endS = text + 39; /* Don't worry about this */


  printf ("The string now says: \"%s\n", text);

  *endS = '\0';
  printf ("The string now says: \"%s\n", text);

// Repeat the following three lines in your own example, but experiment
// with different values than just subtracting 1 from endS

  endS = endS - 1;
  *endS = '\0';
  printf ("The string now says: \"%s\n", text);

  return 0;
}
```

1. Copy the above into a file, save it and compile it and run it. What happens, and why?
2. Try to set the null-character in other positions of the string and print the result (copying the three lines starting with "`endS = endS - 1`"). Make sure you both decrement and increment `endS`. Try using both the normal addition/subtraction and the prefix versions (`++` and `--`).
3. Document what happens and why? Use screenshots to document.



# Exercise 3.3 Calling functions, pass by value, pass by reference

This exercise will help you excel in creating new functions that takes parameters by value and return values.

1. Implement a program with a main function and two other functions:
   - `power(int x, int y)` – should return "x to the power of y", for instance `power(2,3)=8`
   - `multiSwap` – should take three integer-parameters x, y and z. After a call to `multiSwap`, x should be equal to the old y, y should be equal to the old z and z should be equal to the old x. The return type of `multiSwap` should be void.
2. Document your program using inline comments ( `/* … */`)

# Exercise 3.4 Working more with strings

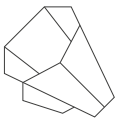
This exercise demonstrates you how you manipulate strings (character arrays); compute length of a string, as well as working with loops.

Based on the below skeleton, complete the program to do the following:

1. Converts each line being input to upper case and print it
2. Computes the length of each line being input and prints the length
3. Possibly combine the above two

Filename: main.c

```c
#include <stdio.h>
#include "my_string_func.h"

#define MAX_LENGTH 1000
int main()
{
     char line[MAX_LENGTH];
     char upper[MAX_LENGTH];
     int line_length;

     /* Use a while loop to read input lines as long as there are any.
        For each input line, print the length of the line as well as
        the upper case version of the line
        NOTE You are NOT allowed to use the standard toupper(…)
        function in string.h
     */
     while ( /* Something should go here */) {

          /* And here do the my_to_upper(…)*/

          printf("Length: %d\t%s\n", line_length, upper);
     }

     return 0;
}
```

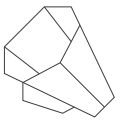


Filename: my_string_func.h

```
#pragma once
/* read_line: read a line into s, return length */
int read_line(char s[]);

/* my_to_upper: Makes an upper-case converted version of str_in in str_out  */
void my_to_upper(char* str_in, char* str_out);
```

Filename: my_string_func.c

```
#include <stdio.h>
#include "my_string_func.h"

/* read_line: read a line into s, return length */
int read_line(char s[])
{
    int c = 0;
    int i = 0;
    int done = 0;

    do
    {
        c = getchar();
        if (c == EOF || c == '\n')
        {
            done = 1;
        }
        else
        {
            s[i++] = c;
        }

    } while (!done);


    s[i] = '\0';
    return i;
}

void my_to_upper(char* str_in, char* str_out)
{

/* You must implement your own version of toupper using the two strings
   str_in and str_out.
   Loop over the str_in (as long as it "contains" something).
   Look at the "current" character and check if its a lowercase letter.
   If it is, convert it to the upper case version and add that to the
   str_out character array
*/
}
```



Filename: my_text_file.txt - Use as test input to your program

```
CALI1
Second_line
ThirdLine
#
LASTLINE_HAS_26_characters
```

4. Implement the function my_`to_upper` and test it on a simple string such as "Hello World!"
5. Implement the main loop to print the line length and the upper-cased version of the input lines – use the file from above as your input file

Note, that there are two ways to "feed" characters/lines of characters to your program. One is to just type using the keyboard and terminate the input feed by pressing 'Enter' on an empty line. Another is to redirect the standard input (keyboard) to a text file. Assume that your executable program is called "myprog.exe" then you redirect the keyboard to a file "myTextFile.txt" as follows:

```
myprog.exe < my_text_file.txt
```

Document your findings!