4. EXERCISES

These are recommended exercises. They are not mandatory and you could work on other exercises if you like. There is a wealth of information out there. The main point of this section is to help you get a minimum of programming experience under your belt.

4.1. Chapter 1:

Write C programs to perform the following tasks.

Exercise 1.1
Input two numbers and work out the difference, average and sum of the squares of the numbers.

Exercise 1.2
Write a program to read a "float" representing a number of degrees Celsius, and print (also as a "float") the equivalent temperature in degrees Fahrenheit. Print your results in a form such as *100.0 degrees Celsius converts to 212.0 degrees Fahrenheit.*

Exercise 1.3
Write a program which will ask you about your name, student number and favourite colour and print those on separate lines. You may use either several printf instructions, each with a newline character in it, or one printf with several newlines in the control string.
Note: Your name is an array of characters. We will look at arrays later on but I will give you an example of a declaration of a char array of 10 letters just for the purpose of this exercise `char a[10]`[§]. Don't forget to use the correct type specifier for a string of characters in your i/o functions.

Exercise 1.4
Write a program to read a number of units of length (a float) and print out the area of a circle of that radius. Assume that the value of pi is 3.14159.
Your output should take the form: The area of a circle of radius ... units is .... units.
If you want to be clever, and have looked ahead in the notes, print the message "Error: Negative values not permitted." if the input value is negative.

Exercise 1.5
Given as input an integer number of seconds, print as output the equivalent time in hours, minutes and seconds. Recommended output format is something like
7322 seconds is equivalent to 2 hours 2 minutes 2 seconds

---

§ I did mention that `scanf` can do silly things when dealing with inputting spaces or characters/strings terminated with "enter". This should not concern us too much at this time. If you are encountering problems, for now, consider the below scanf formulations as hints for you go through your exercises (it's ok if you do not fully understand them).

```
scanf(" %c", &c);
\\ to input a single character e.g. char c. Note the space before %c


scanf("%[^\n]s", &c[0]);
\\ if you enter a string e.g. char c[10] terminated by "enter"
```

Exercise 1.6
Write a program to read two integers with the following significance.
The first integer value represents a time of day on a 24 hour clock, so that 1245 represents quarter to one mid-day, for example.
The second integer represents a time duration in a similar way, so that 345 represents three hours and 45 minutes.
This duration is to be added to the first time, and the result printed out in the same notation, in this case 1630 which is the time 3 hours and 45 minutes after 12.45.
Typical output might be e.g., Start time is 1415. Duration is 50. End time is 1505.
Start time is 2300. Duration is 200. End time is 100.

You are not allowed to use `if... else` statements to solve this exercise. You can however, use expressions containing comparison operator, e.g., the expression `x_gr_zero=(x>0)*x`; yields x for x>0 and 0 otherwise.

4.2. Chapter 2:

Exercise 2.1
Write a program to read two characters, and print their decimal value when interpreted as a 2-digit hexadecimal number. Accept upper case letters for values from 10 to 15. Return an error for invalid entries.

Exercise 2.2
Read an integer value. Assume it is the number of a month of the year; print out the name of that month. If the number is invalid, print an error message. Write a program that repeats this for 3 month entries.

Exercise 2.3
Write a program to read in 10 numbers and compute the average, maximum and minimum values. Finally, sort them in order to get an output something like (only 5 numbers shown): (*hint for printing to screen:* just as \n means a new line, \t introduces a tabulation)

```
You entered       The sorted list is
   10                   1
   15                   2
   2                    10
   1                    15
   43                   43
```

Exercise 2.4
Write a program to read in numbers until the number -999 is encountered. The sum of all number read until this point should be printed out. *TIP*: you don't need to use arrays for this exercise.

Exercise 2.5
Read in three values representing respectively: 1) a capital sum (**integer** number of dollars), 2) a rate of interest in percent (float), and 3) a number of years (integer).
Compute the values of the capital sum with compound interest added over the given period of years.
Each year's interest is calculated as
interest = capital * interest_rate / 100;
and is added to the capital sum by capital
+= interest;

Print out money values in dollars accurate to two decimal places. (*hint:* try and use output format %.*n*f to get *n* decimal places)
Print out a floating value for the value with compound interest for each year up to the end of the period.
Print output year by year in a form such as:

```
Original sum 30000.00 at  12.5 percent for 10 years

      Year Interest  Sum
      ----+-------+--------
      1  3750.00 33750.00
      2  4218.75 37968.75
      3  4746.09 42714.84
      4  5339.35 48054.19
      5  6006.77 54060.96
      6  6757.62 60818.58
      7  7602.32 68420.90
      8  8552.61 76973.51
      9  9621.68 86595.19
      10 10824.39 97419.58
```

Exercise 2.6
Read a positive integer value, and compute the following sequence: If the number is even, halve it; if it's odd, multiply by 3 and add 1. Repeat this process until the value is 1, printing out each value.
Finally print out how many of these operations you performed.
Typical output might be:

```
Initial value is  10
Next value is   5
Next value is  16
Next value is   8
Next value is   4
Next value is   2
Final value 1, number of steps 5
```

If the input value is less than 1, print a message containing the word `Error`

Exercise 2.7
Write a function that asks for an input n and outputs a right-side-up triangle of height *n* and width *2n1*; the output for *n* = *6* would be:

```
     *
    ***
   *****
  *******
 *********
***********
```

Exercise 2.8
Write a program which asks you to input your student number (XXXXXXX) and then presents you with a menu for the following operations (by inputting a number the corresponding activity is executed):

1. Write the number back to you in reverse order
2. Asks you to state a digit number *n*, then prints the n^{th} character of the student number
3. Asks you for a particular character and tells you if it is found in the student number
4. Quit

After each operation (except 'quit'), the program goes back to the main menu.

4.3. Chapter 3:

Exercise 3.1

Write a function to check if an integer is negative; the declaration should look like `int is_positive(int i);` The program will keep asking for inputs and will keep delivering outputs (informing the user of whether the number is negative or not) until the number entered is 999 (such input will force quit).

Exercise 3.2

Write a function with two arguments to raise a floating point number to an integer power, so for example:

```
float a = raise_to_power(2, 3); //a gets 8 float
b = raise_to_power(9, 2); //b gets 81
float raise_to_power(float f, int power); //make this your declaration
```

The program will keep asking for inputs and will keep delivering outputs (informing the user of the computed result) until the numbers entered are 999, 999 (such input will force quit).

Note: you are not allowed to use the function `pow` here (elevating to power) but should rather use iterative multiplication to compute the result.

Exercise 3.3

Write a function which returns 1 if a positive integer is prime and 0 otherwise. A prime number can only be divided (without remainder) by itself and by 1. (*hint*, the modulo operator % might be helpful for your function)

Exercise 3.4

Write a function to determine the number and identity of all prime numbers below an inputted positive number *n*. To check operation, try at least two separate runs for values of *n* in excess of 41. *Hint:* have you solved the exercise above?

Exercise 3.5

Write a program to calculate the factorial of an integer number using recursion (hint: very similar to the example in the concepts section). Remember that 3!=3*2*1=6 and 0!=1. Create suitably explanatory inputs and outputs.

Exercise 3.6

Write a function called half() that takes an integer argument. The function must print the number it received to the screen, then the program should divide that number by two to make a new number. If the new number is greater than zero the function then calls the function half() (itself!) passing it the new number as its argument. If the number is zero or less than zero the function exits.

So, for example, calling the function half() with an argument of 100, the screen output should be

```
100
50
25
...
...
1
```

4.4. Chapter 4:

Exercise 4.1

Write a C program that asks the user for an input and stores it in a variable. Then, the program prints out something like.

```
 The value entered is <value> and is stored at memory location <location
value>.
```

Tip: use the `%p` type identifier to visualise the memory address (pointer value) in hexadecimal format.

Exercise 4.2
Write a program that takes three variable (a, b, c) in as separate parameters and calls on a function which rotates the values stored so that value a goes to b, b to c and c to a. Output the values of your variables before and after the swap in an understandable manner.

Exercise 4.3
Write a program that creates an array of type *float*. How you populate this array is up to you. Print out the content of the array and the corresponding memory addresses for each entry. (you may want to try with both equivalent ways of referencing an array to make sure your syntax is up to speed). Take note of the numeric difference between successive address entries. What does that suggest regarding the way the array is stored to memory? What is the size of a *float* entry?

Exercise 4.4
Write a C program to read through an array of type *int* using pointers. How you populate this array is up to you. Write the array to screen so that it fits in one screenshot. Then ask the user to input a value. Your C program is to scan through the array to find the value and output how many occurrences were found. You must use pointers.

Exercise 4.5
Write a program to find the number of times that a given word (i.e. a short string) occurs in a sentence (i.e. a long string!). Read data from standard *scanf* input (you will store two arrays of *char* to memory). You will first acquire a single word, and then a longer string of general text. Your program should pass the two strings to a function which will count the number of occurrences. ***Extension:*** Let the function also turn any blank spaces into dashes.

Typical output should be something like:

```
The word is "the".
The sentence is "the cat sat on the mat".
The word occurs 2 times.
If I remove the spaces: "the-cat-sat-on-the-mat"
```

Important Tip: *when you type a space scanf will think that your string is finished. The syntax:* `scanf("%[^\n]s", c);`, *where c will be your string array, will ignore the spaces and capture the string as a whole once the enter key is pressed (it sets \n as the string delimiter). By the way, note that the variable is passed as c and not &c, you should be able to tell why.*

4.5. Chapter 5:

Exercise 5.1
Implement, using structures and functions as appropriate, the sum and product of two fractions. Simplification is not required. To get full marks, you should pass your structures to your functions as pointers.

Exercise 5.2
Implement, using structures and functions as appropriate, a program which requires you to enter a number of points in 3 dimensions. The points will have a name (one alphanumeric character) and three coordinates x, y, and z. Find and implement a suitable way to stop the input loop. The program, through an appropriate distance function, should identify the two points which are the furthest apart. Another function should calculate the centre of gravity of the point cloud (i.e., the average of each coordinate for all points entered)

The output should show a list of the points entered, then name and list the two that are furthest apart, and finally list the centre of gravity.

Exercise 5.3

Create a C program which describes a store management software. The records will be saved in a structure called goods. You can have up to 10 goods in your store (array of struct). Each good will have a name, a record id (sequential numbering), a price, and a flag as to whether it is in stock or not. Use appropriate variable types for each.

Your program will present a menu, allowing the user to choose:

```
1. Open record (by number)
2. List all records
3. Quit
```

Quit is self-explanatory. List all records will print out a list of record IDs and product names.

If Open record is selected, then the program should show

```
Record <number> opened

<a list of the details of the item should appear here>
```

And give the following options:

```
1. Change item name
2. Change item price
3. Change stock availability
4. Go back
```

After a suboption has been chosen, the user should have the opportunity to enter the new name/price or toggle availability. Then the program should show the updated details of the record and show the second menu for a new choice.

Create a suitable main program structure which calls on functions which will carry out the operations as described, asking for appropriate inputs and returning appropriate outputs.

Note: you should use pointers when passing structs to functions.