Course: DD2325 - Exercise Set 4

Exercise 1: *Basic Pointer Manipulation*

Below is a simple `C` program using pointers. What will this program print to the screen?

```
#include <stdio.h>

main()
{
   int *p1, *p2;
   int v, v1;

   p1 = &v;
   *p1 = 42;
   p2 = p1;
   printf("*p1 == %d, *p2 == %d\n", *p1, *p2);

   *p2 = 53;
   printf("*p1 == %d, *p2 == %d\n", *p1, *p2);

   p1 = &v1;
   *p1 = 88;
   printf("*p1 == %d, *p2 == %d\n", *p1, *p2);

   printf("I hope you get the point of this example\n");
}
```

Remember! A *pointer* is the memory address of a variable. Recall the computer's memory is divided into numbered memory locations, called *bytes* and variables are stored as a sequence of adjacent memory locations.

Exercise 2: *Pointers - passing arguments by reference*

Perform the following tasks:

1. Write a function that computes the roots of a quadratic equation and then returns the value of these roots via the arguments of the function. Remember to check if the roots are real. If they are not send an error message.

2. Write a function that swaps the value of two numbers if the first is

larger than the second.

3. Complete the tasks specified in the `main()` loop below.

```
#include <stdio.h>

main()
{
   double a=2.0, b=10.0, c=9.0;
   double x1, x2;

   /* call your root finding function and */
   /* store the answers in x1 and x2 */

   /* Ensure x1 holds the value of the smaller root */

   /* Print out the roots */
}
```

Exercise 3: *Pointers - passing arguments by reference*

The area of a triangle with sides of length a, b and c can be computed from Heron's formula:

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

where s is the semiperimeter

$$\text{s} = \frac{1}{2}(a+b+c)$$

Write a function that uses five parameters: three value parameters that provide the lengths of the triangle sides, and returns the area and perimeter via parameters passed by reference. Note that not all combinations of a, b and c produce a traingle. Write a function that checks if a, b and c correspond to a valid triangle.

Exercise 4: *Arrays as input to functions*

Write a function called `InOrder` that returns an integer. It takes as parameters an array of `double`s and an integer which is the length of the input array. This function will test if the input array is sorted in ascending order, that is given `a` as the input array then

```
a[0] <= a[1] <= a[2] <= ....
```

The function returns `1` if the elements are in order, otherwise it will `0`.

Exercise 5: *Arrays as input to functions*

Write a program that reads in a line of text and outputs a list of all the letters that occur in the text together with the number of times each letter occurs. You can assume that only letters and whitespace are input. So for example the input:

```
do be do bo
```

should produce output similar to the following:

```
d:2, o:3, b:2, e:1
```

Help this snippet of code shows you how you can read in a line of text from standard input (ie the terminal window).

```
#include <stdio.h>

const int MAX_STR_LEN = 400;

main()
{
   char str[MAX_STR_LEN];
   printf("Enter a line of text:\n");
   fgets(str, sizeof(str), stdin);
   printf("You have entered:\n %s", str);
}
```

Exercise 6: *Dynamic arrays*

Complete the program below. It reads in `n` (variable) integer values from the user and stores these integer values in the dynamic array `numbers` whose size is updated after reading in each new number.

```
#include <stdio.h>
#include <stdlib.h>

main ()
{
   int input,n;
   int count=0;
   int * numbers = NULL;
```

```
  do {
     printf("Enter an integer value (0 to end):  ");
     scanf("%d", &input);
     count++;
     /* Your input goes here */

  } while (input!=0);

  printf ("Numbers entered:  ");
  for (n=0;n<count;n++) printf("%d ", numbers[n]);
  free(numbers);
}
```

Exercise 7: *Structures*

A rational number is a number that can be represented as the quotient of two integers, that is $\frac{a}{b}$ where a and b are integers. Write a rational number `struct`. Then write the following functions:

- `my_rational add_rationals(my_rational r1, my_rational r2)`
- `my_rational multiply_rationals(my_rational r1, my_rational r2)`
- `int rationals_equal(my_rational r1, my_rational r2)`
- `my_rational normalize_rational(my_rational r)`

The latter function returns a rational number where the denominator is positive and the numerator and denominator are as small as possible. For example, after normalization `4/-8` should be `-1/2`.

Exercise 8: *Linked List*

You will write a simple linked list example. This will require writing the following functions and structures.

- Write a struct called `node` that will be used in a linked list containing integers.
- Given an `int` and a reference to the head pointer (i.e. a `struct node**` pointer to the head pointer), add a new node at the head of the list.
- Write a function which returns the number of nodes in the list.

- Write a `main` loop that reads in the list {`1, 2, 3`}.

Exercise 9: *Linked List*

Building on the first linked list example, write a `Count` function that counts the number of times a given int value occurs in a list. The code for this has the classic list traversal structure.

Exercise 10: *Linked List*

Building on the first linked list example, write a function `GetNth` that takes a linked list and an integer index and returns the data value stored in the node at that index position. `GetNth()` uses the `C` numbering convention that the first node is index 0, the second is index 1, ... and so on.

Exercise 11: *File opening, reading and closing*

For the lab session you have to read data from a text file. I have written out an example problem reading data from a file which we will examine together during the exercise class.