COMP10002 Workshop Week 3

Outlook: *Now: login into your laptop or lab's PC and be ready with* `jEdit` *and* `minGW` *(or* `Terminal`*). Also start your web browser.*

1	Questions…
2	Discuss: Ex 4.2 and 4.1
3	Functions: Do it together Ex 4.5 Also learn: using
4	Discuss: HowTo and Tools for Ex 4.6 & 4.7
5	Lab: Implement 4.6 + 4.7, 5.6
github this week	- frame.c - skeleton and sample data for exercise 4.5 - skeleton and sample data for exercise 5.6 - sample data for exercise 4.6 & 4.7

Do it together, and now:

Start jEdit and minGW (or MacBook's Terminal)
In minGW window, create folder week3:

```
cd H:
cd comp10002
mkdir week3
cd week3
```

On your web browser, naviate to
git@github.com/anhvir/c102 , here you can get some program skeletons and instructions.

Not now, but later: Learn Redirection: after finishing grapher, try:

```
grapher < grapher.data >grapher.out
cat grapher.out
```

What learnt in week 2, questions?

loop?

function?

recursive function?

Exercise 4.2

Give a general construction that shows how any `do` *statement can be converted into an equivalent* `while` *statement.*

4.1 a)

Trace the action of the loop, and determine the values printed out by the `printf` *statement. Assume that all variables have been declared to be of type* `int`

```
1  for  (i=0; i<20; i= i+3) {
2     printf ("%2d\n", i);
3  }
```

4.1 b-c

```
b1   for  (i=1; i<2000000; i= 2*i) {
b2        printf ("%7d\n", i);
b3   }

c1   sum = 0;
c2   for  (i=0; i<10; i++) {
c3        sum = sum + i;
c4        printf ("S(%2d) = %2d\n", i, sum);
c5   }
```

4.1 d-e

```
d1  for (i= 0; i < 8; i++) {
d2    for  (j= i+1; j < 8; j += 3) {
d3      printf ("i= %d, j= %d\n", i, j);
d4    }
d5  }

e1  for (i= 0; i < 8; i++) {
e2    for  (j= i+1; j < 8; j += 3) {
e3      if (i+j == 7) {
e4        break;
e5      }
e6      printf ("i= %d, j= %d\n", i, j);
e7    }
e8  }
```

4.1 f

```
f1   j = 5;
f2   for (i= 0; i < j; i++) ; {
f3       printf (“i= %d, j= %d\n”, i, j);
f4   }
```

4.1 f-g

```
f1   j = 5;
f2   for (i= 0; i < j; i++) ; {
f3       printf (“i= %d, j= %d\n”, i, j);
f4   }


g1   j = 5;
g2   for (i= 0; i < j; j++) {
g3       printf (“i= %d, j= %d\n”, i, j);
g4   }
```

Write a function that computes:

a) $n!$

b) $1/1^2 + 1/2^2 + \ldots + 1/n^2$

c) $1 + x + x^2/2! + x^3/3! + \ldots + x^n/n!$ where n is the smallest positive integer that satisfies

$$| x^n/n! | < 10^{-6}$$

4.5 – Design (Discussion)

Design and implement a program `grapher.c` *that reads integers and draw a simple graph. Assume that all of the values read are between 1 and 70. Example:*

```
H: grapher
Enter integers between 1 and 70 inclusive: 3 7 11
  3 |***
  7 |*******
 11 |***********
```

4.5 – Design (Discussion)

```
H: grapher
Enter integers between 1 and 70 inclusive: 3 7 11
  3 |***
  7 |*******
 11 |***********
```

4.6, 4.7 – Design & Tools (Discussion)

Design a program `my_wc` *that count the number of characters, words, and lines in the input. Example of execution:*

```
H: my_wc
Enter text:
Mary has a little lamb,
Little lamb, little lamb;
^D    (or ^Z if using MinGW/Windows)
Lines: 2
Words: 9
Chars: 26
```

4.6, 4.7 – Design & Tools (Discussion)

How to read one character?
How to recognize the end of input?
How to know that it's end of a line?
How to know that it's within a "word" or not? How to define a "word"?
What is a possible algorithm for the task?

Lab: Implement 4.5, 4.6+4.7, 5.6

4.5: *Design a program* `grapher.c` *that reads integers and draw a simple graph. Graph example:*

```
 3 |***
11 |***********
```

4.6+4.7: *Design a program* `my_wc` *that count the number of characters, words, and lines in the input. Use* `mary.txt` *to test your program, that is:*

```
bash $ my_wc < mary.txt
```

5.6: Two numbers are an amicable pair if their factors (excluding themselves) add up to each other. The first such pair is 220, which has the factors [1, 2, 4, 5, 10, 11, 20, 22, 44,55, 110], adding to 284; and 284, which has the factors [1, 2, 4, 71, 142], the sum of which is 220. The next pairs are 1,184 and 1,210; and then 2,620 and 2,924.

Write a function that takes two int *arguments and return true if they are an amicable pair. Test the function using an appropriate scaffolding [so now you also need main(), of course]. Use* `e56.data` *to test your program.*

CHALLENGE: write a program that search for amicable pairs and print them!