# *COMP10120 Practical Set 8: Unions and Dynamic Data Structures*

**Please read the questions carefully. Name each program based on your student number, the practical set number and question number. For this set (set8), question 1 should be named 1234567s8q1.c where your student number replaces 1234567. All questions that you are submitting can be zipped into a single file called 1234567s8.zip, where 1234567 is your student number and s8 refers to set 8. Please also include a readme.txt file which says which compiler you used to test your implementation. This zipped file can be submitted via Moodle for grading.**

**Part 1**

1. Write a C Program that creates a **Union** with members char a, short b, int c and long d. In the program prompt the user to input values of type char, short, int and long and the store the values in the Union variables. Include code which prints the size of the Union. Each Union variable should also be printed. Examine the output in a comment in the program.

2. Using the structures below, write a C Program which can fill and initialise the structures with user data (read from the files towns.txt and counties.txt) and then print out the name, population, county's longName, county's shortName and the county's population for each of the 5 town records by using struct member reference notation.

```
struct county{
        char *longName;
        char *shortName;
        unsigned int population;
};

struct town{
        char *name;
        unsigned int population;
         struct county aCounty;
};
```

3. Use **dynamic memory allocation** (calloc, realloc) to write a C Program which achieves the following:
   - Prompts the user to enter the type (integer or float) and number of elements they wish to store in an array
   - *Create* sufficient memory to store the array
   - Ask the user to enter each of the values they want to store
   - After all elements have been entered and stored in the array, calculate the average and print to the screen.
   - Ask the user if there are **more** elements to be added to the array, if yes, prompt the user to enter the number of additional elements.
   - *Reallocate* sufficient memory to the array to store the new number of elements
   - Ask the user to enter each of the new values they want to store
   - After all the new elements have been entered and stored in the array, calculate the average and print to the screen.