COMP1511 18s2

# Week-03 Tutorial Exercises

1. Tuts from now on will start with **code reviews**. During the tutorial this week, your tutor will show you how to conduct a code review on the following code:

   - **confusing.c**

   **Your Task**: Please download the above code and **read it before** you come to your tutorial. You need to figure out how the code works and what it does. In your tutorial you can discuss how it can be improved to make it more readable and to make it clearer.

   Your tutor will nominate a lab pair this week to do next week's code review.

   How a code review works:

   - The reviewees show their code on the projector, and briefly walk through how it works.
   - The class and tutor (the reviewers) give feedback, ask questions, make suggestions. You tutor will show you how to do this at first but then will expect the other reviewers to take over.
   - Reviewers are NOT negative, a review is to be supportive and constructive and helpful to the reviewees.
   - The reviewees should speak very little, just give a brief overview of the code they want reviewed.
   - Let everyone have a turn to speak, donâ€™t dominate the conversation.
   - Contribute and participate, donâ€™t be silent. If you donâ€™t understand anything then that doesnâ€™t mean be silent - it means ASK for an explanation. By asking you are helping the coders to see how to be clearer.

2. Which of the following are valid variable names in C?
   If they are valid, would they be a good name?

   - `THX1138`
   - `2for1`
   - `mr_bean`
   - `my space`
   - `event_counter`
   - `^oo^`
   - `_MEMLIMIT`
   - `return`

3. C is a *typed* language. What does this mean? Why is this useful?

4. What is output by the following C program? Why? Make sure you compile the program and run it to confirm your answer.

```
#include <stdio.h>

#define FIRST_NUMBER     10
#define SECOND_NUMBER    20
#define TOTAL   FIRST_NUMBER + SECOND_NUMBER
#define AVERAGE TOTAL / 2

int main(void) {
    printf("The average of %d and %d is %d\n",
           FIRST_NUMBER, SECOND_NUMBER, AVERAGE);

    return 0;
}
```

5. The value of CÂ  arithmetic operations depends on the types of the operand. If the types of the operands differ, CÂ automatically converts the types as appropriate.
   Determine the value of each expression and sub-expression:

   i. `1 / 2 * 500`
   ii. `1 / 2.0 * 500`
   iii. `(17 / 5) * 5 + (17 % 5)`
   iv. `(12 - 17) % 6 - 4`

6. In your lab this week you will be implementing a function that determines whether a given year is a **leap year** or not. In the tutorial, discuss how you might draw the control flow for determining whether a year is a leap year or not.

7. Write a C program `multiple_of_ten.c` which reads 2 integers and then prints all of the multiples of ten between those numbers.

```
$ ./multiple_of_ten
Enter start: 12
Enter finish: 42
20
30
40
$ ./multiple_of_ten
Enter start: -1
Enter finish: 7
0
```

8. Discuss the errors in these **while** loops

```
int i;

while (i < 100) {
    printf("%d\n", i);
    i = i + 1;
}
```

```
int i = 0;
int j = 0;

while (j = 1 || i < 100) {
    printf("%d\n", i);
    i = i + 1;
}
```

```
int i = 0;
int n = 10;
while (i < n) {
    printf("%d\n", i);
    n = n + i;
    i = i + 1;
}
```

```
int i = 0;
while (i < 10)
    printf("%d\n", i);
    i = i + 1;
```

9. Write a program that reads in an integer and prints out that many asterisks, each on a new line.

```
$ ./asterisks
Please enter an integer: 5
*
*
*
*
*
```

10. Consider the following program square.c

```
#include <stdio.h>

int main(void) {
    int number;
    int row, column;

    // Obtain input
    printf("Enter size: ");
    scanf("%d", &number);

    row = 1;
    while (row <= number) {
        column = 1;
        while (column <= number) {
            printf("*");
            column = column + 1;
        }
        printf("\n");
        row = row + 1;
    }

    return 0;
}
```

The output if the user types in the number 5 is:

```
$ ./square
Enter size: 5
*****
*****
*****
*****
*****
```

Modify the program so that it prints out a triangle like this:

```
$ ./triangle
Enter number: 5
----*
---**
--***
-****
*****
```

Now modify so it prints the following pattern:

```
$ ./diagonal
Enter an integer: 10
*---------
-*--------
--*-------
---*------
----*-----
-----*----
------*---
-------*--
--------*-
---------*
```

Now modify so it prints the following pattern:

```
$ ./bars
Enter an integer: 9
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
_*_*_*_*_
```

## Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session.
Your tutor may still choose to cover some of the questions time permitting.

11. Write a program that reads two integers (height x length) and prints a rectangular, asterisk outline with the specified dimensions, for example:

```
$ ./rectangle
Enter rectangle height and length: 3 5
*****
*   *
*****
```

12. For the program above consider and discuss the ways in which you would test their correct behaviour. Then extend the program to include error checking for, and handling of, invalid input.

13. Design and write a program that reads an integer *n* and prints a diamond asterisk outline with side length *n*, for example:

```
$ ./diamond
Enter side length: 3
  *
 * *
*   *
 * *
  *
$ ./diamond
Enter side length: 6
     *
    * *
   *   *
  *     *
 *       *
*         *
 *       *
  *     *
   *   *
    * *
     *
```

14. This is a revision question, to make the difference between *if* and *while* statements clear. Consider the two code snippets below, what will be the output after execution of each of the programs for *input_num = 3*.

```
int input_num, i;

printf("Enter a number: ");
scanf("%d", &input_num);

i = input_num;

if (i <= 5) {
  printf("%d\n", i * i);
  i++;
}
```

```
int input_num, i;

printf("Enter a number: ");
scanf("%d", &input_num);

i = input_num;

while (i <= 5) {
  printf("%d\n", i * i);
  i++;
}
```

15. Write a C program `decompose.c` that prompts the user to enter an integer, reads it from the input and prints out the number in individual digits. Allow the program to work for input numbers up to 5 digits, i.e. up to 99999. You should be able to write this program using some divisions and remainder (modulo '%') operations, `if` statements and simple comparisons.
Your program should behave as follows:

```
$ ./decompose
Please enter an integer:
25
You entered 25 which is decomposed: 2 5
$ ./decompose
Please enter an integer:
2825
You entered 2825 which is decomposed: 2 8 2 5
$ ./decompose
Please enter an integer:
2
You entered 2 which is decomposed: 2
```

Your program should handle all integers in range (0 to 99999).

Hint use `if` divide (`/`) and mod (`%`).