Week 01 Tutorial

Getting Started and Recap!

More questions are coming ...

1. Class introduction (for everyone, starting with the tutor):
   - introduce yourself to the other people in the tute class
   - discuss what you learned from COMP1511
   - discuss what parts you're still unsure of
   - choose lab partners (who you'll work with until week 5)

2. Consider the Coding Style Guide from COMP1511.

   a. What is the purpose of such a style guide?
   b. What are some advantages of relaxing some of the restrictions in COMP2521?
   c. What are some disadvantages of relaxing some of the restrictions in COMP2521?

   Think about your answers to this questions as you answer the following questions.

3. A C program has several means of interacting with its run-time environment. Describe what each of the following is and what it is used for: `argc`, `argv`, `stdin`, `stdout`, `stderr`, `exit()`.

4. Consider a program called `myprog` which is invoked as:

```
$ ./myprog  hello there,  'John Shepherd'  >  myFile
```

   a. What are the values of `argc` and `argv`?

   b. Where would the statement `printf("%s",argv[1])` place its output?

   c. Where would the statement `ch = getchar();` get its input?

5. Which of the following statements are equivalent? Which are incorrect? Assume that `x` is an `int` variable.

   a. `scanf("%d", x);`

   b. `scanf("%d", &x);`

   c. `printf("%d", x);`

   d. `fscanf(stdin, "%d", &x);`

   e. `fscanf(stderr, "%d", &x);`

   f. `fprintf(stdout, "%d", x);`

   g. `fprintf(stderr, "%d", x);`

6. What is the effect of the following assignment statement?

```
int a; int b; int c; int d; int e;
a = b = c = d = e = 0;
```

   Would it make any difference if the assignments were not done?

7. Consider the following two sets of definitions and statements:

```
int   x, y;
char *c, *d, *e, *f;

x = y = 2;
c = d = "abc";
e = "xyz"; f = "xyz";

x++;
*c = 'A';
```

   a. Show the state of the memory after the first set of assignments.
   b. Would the be different if we had done `c = "abc"; d = c;`?
   c. Show the state of memory after `x++;`.
   d. What happens when `*c = 'A';` is executed? Why?

8. Describe the difference in behaviour (and in the final result) of the following two `switch` statements:

```
// S1                                           // S2
switch (ch) {                                   switch (ch) {
case 'a':  printf("eh? "); break;               case 'a':  printf("eh? ");
case 'e':  printf("eee "); break;               case 'e':  printf("eee ");
case 'i':  printf("eye "); break;               case 'i':  printf("eye ");
case 'o':  printf("ohh "); break;               case 'o':  printf("ohh ");
case 'u':  printf("you "); break;               case 'u':  printf("you ");
}                                               }
printf("\n");                                   printf("\n");
```

   What will be printed for each of the following values for `ch`: `'u'`, `'o'`, `'e'`?

9. Consider the following function to convert a number in the range 0..6 into a weekday name. 0 returns "Sun", 1 returns "Mon", 2 returns "Tue", and so on.

```
char *numToDay(int n)
{
    assert(0 <= n && n <= 6);
    char *day;
    if (n == 0) {
        day = "Sun";
    } else if (n == 1) {
        day = "Mon";
    } else if (n == 2) {
        day = "Tue";
    } else if (n == 3) {
        day = "Wed";
    } else if (n == 4) {
        day = "Thu";
    } else if (n == 5) {
        day = "Fri";
    } else if (n == 6) {
        day = "Sat";
    }
    return day;
}
```

Suggest at least two alternative and more concise ways of achieving the same result. Include the `assert(...)` in each case.

For each function, including the one above, explain what would happen if the `assert(...)` was removed and the function was invoked via `numToDay(7)`.

10. Give an `if` statement equivalent to the following `switch` statement:

```
assert(islower(ch));
switch (ch) {
case 'a':
case 'e':
case 'i':
case 'o':
case 'u':
    printf("vowel"); break;
default:
    printf("consonant"); break;
}
```

11. Use a conditional expression to replace the `if` in the following code:

```
ch = getchar();
if (isdigit(ch))
    type = "digit";
else
    type = "non-digit";
printf("'%c' is a %s\n", ch, type);
```

How should each of the variables (`ch` and `type`) be declared?

12. What `while` loop is equivalent to the following `for` statement:

```
for (ch = getchar(); ch != EOF; ch = getchar()) {
    putchar(ch);
}
```

13. Consider the following program (adapted from Sedgewick):

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

int main(int argc, char *argv[])
{
    int i, j, *a;
    int N = 0;

    // initialisation
    assert(argc > 1);
    sscanf(argv[1], "%d", &N);
    assert(N > 0);
    a = malloc(N*sizeof(int));
    assert(a != NULL);
    for (i = 2; i < N; i++) a[i] = 1;

    // computation
    for (i = 2; i < N; i++) {
        if (a[i]) {
            for (j = i; i*j < N; j++) a[i*j] = 0;
        }
    }

    // results
    for (i = 2; i < N; i++) {
        if (a[i]) printf("%d\n",i);
    }
    return EXIT_SUCCESS;
}

```

Try to answer each of the following questions about this program:

a. there are no braces around the bodies of some `for` loops; does this matter?
b. what is the line of code `sscanf(argv[1],"%d",&N);` doing?
c. suggest an alternative for the `sscanf(...)` statement?
d. for each of the `assert`s ...
    - describe what error is being checked for and why
    - suggest a better error message than what you get from `assert`

   e. what are the values of `a[0]` and `a[1]` during execution?
   f. why don't the values of `a[0]` and `a[1]` matter?
   g. what is the purpose of this program?

14. Define a `swap()` function that exchanges two elements in an array.

    For an array `a[]` and two indexes `i` and `j`, we could exchange the *i*'th and *j*'th elements by the call:

    ```
    swap(a, i, j)
    ```

15. Consider a function to check whether the elements in an array occur in ascending order. Duplicates are allowed in the array, as long as they are adjacent.

    The condition we are testing for can be stated more formally as a post-condition on the function as below:

    ```
    // Pre:
    // - a[] is a valid pointer to the start of an array of ints
    // - n is a positive int indicating how many elements in a[]
    // Post:
    // - return value = ∀ i ∈ {0..n-2} ( a[i] ≤ a[i+1] )
    bool isSorted(int *a, int n)
    {
        ...
    }
    ```

    Implement this function in two styles:

    a. using COMP1511 C Style
    b. using for, break/return to exit the loop early

16. Consider the following simple linked-list representation and a function that sums the values in the list:

    ```
    typedef struct _Node {
        int value;
        struct _Node *next;
    } Node;

    typedef Node *List;  // pointer to first Node
    ```

    Write a function to sum the values in the list. Implement it first using `while` and then using `for`. Finally, implement it using recursion.

17. **Important:** Make sure that you properly understand and can effectively use the following mechanisms (covered in COMP1511). In case you have any questions, please discuss them with your tutor.

    - Structs (see videos on Structs)
    - Pointers (see videos on Pointers)

- Malloc (see videos on Malloc)
- Linked Lists (see videos on Linked Lists)
