Week 3 Assessment Questions

1. (Dynamic linked lists)

   Write a C-program called **llcreate.c** that creates a linked list of integers from user input. Your program should use the following functions:

   - *void freeLL(NodeT *list)*: taken from the lecture.
   - *void showLL(NodeT *list)*: taken from the lecture **but needs modification.**
   - *NodeT *joinLL(NodeT *list, int v)*: returns a pointer to the linked list obtained by appending a new element with data *v* at the end of *list*. **Needs to be implemented.**

   The program:

   - starts with an empty linked list called `all` (say), initialised to `NULL`
   - prompts the user with the message "Enter a number: "
   - appends at the end of `all` a new linked list node created from user's response
   - asks for more user input and repeats the cycle
   - the cycle is terminated when the user enters any non-numeric character
   - on termination, the program generates the message "Done. List is " followed by the contents of the linked list in the format shown below.

   A sample interaction is:

   ```
   prompt$ ./llcreate
   Enter an integer: 12
   Enter an integer: 34
   Enter an integer: 56
   Enter an integer: quit
   Done. List is 12-->34-->56
   ```

   Note that any non-numeric data 'finishes' the interaction. If the user provides no data, then no list should be output:

   ```
   prompt$ ./llcreate
   Enter an integer: #
   Done.
   ```

   We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `llcreate.c` *in the current directory. You can use dryrun as follows:*

   ```
   prompt$ 9024 dryrun llcreate
   ```

   Note: Please ensure that your output follows exactly the format shown above.

2. (Dynamic linked lists)

   Extend the C-program from the previous exercise to split the linked list in two halves and output the result. If the list has an odd number of elements, then the first list should contain one more element than the second.

   Note that:
   - your algorithm should be 'in-place' (so you are not permitted to create a second linked list or use some other data structure such as an array);
   - you should not traverse the list more than once (e.g. to count the number of elements and then restart from the beginning).

   An example of the program executing could be

   ```
   prompt$ ./llsplit
   Enter an integer: 421
   Enter an integer: 456732
   Enter an integer: 321
   Enter an integer: 4
   Enter an integer: 86
   Enter an integer: 89342
   Enter an integer: 9
   Enter an integer: #
   Done. List is 421-->456732-->321-->4-->86-->89342-->9
   First part is 421-->456732-->321-->4
   Second part is 86-->89342-->9
   ```

   To test your program you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `llsplit.c` *in the current directory. You can use dryrun as follows:*

   ```
   prompt$ 9024 dryrun llsplit
   ```

   Note: Please ensure that your output follows exactly the format shown above.

Assessment

Submit your solutions to Questions 1 and 2 using the following **give** command:

```
prompt$ give cs9024 week3 llcreate.c llsplit.c
```

Make sure you spell the filenames correctly. You can run **give** multiple times. Only your last submission will be marked.

The deadline for submission is **Tuesday, 23 June 11:00:00am**.

Each program is worth 1 mark. Total marks = 2. Auto-marking will be run by the lecturer several days after the submission deadline using different test cases than `dryrun` does.

Hints:

- Programs will not be manually marked.
- It is important that the output of your program follows exactly the format shown above, otherwise auto-marking will result in 0 marks.
- **Ensure that your program compiles on a CSE machine with the standard options `-Wall -Werror -std=c11`. Programs that do not compile will receive 0 marks.**
- `dryrun` and auto-marking also check the correct usage of dynamic memory allocation and pointers as well as the absence of memory leaks.
- Do your own testing in addition to running `dryrun`.