Homework Three

Q1. Consider the following function:

```
/* Makes an array of 10 integers and returns a pointer to it */

int *makeArrayOfInts(void) {
  int arr[10];
  int i;
  for (i=0; i<10; i++) {
    arr[i] = i;
  }
  return arr;
}
```

Explain what is wrong with this function. Rewrite the function so that it correctly achieves the intended result using malloc().

Q2. Consider the following program:

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

void func(int *a) {
  a = malloc(sizeof(int));
}

int main(void) {
  int *p;
  func(p);
  *p = 6;
  printf("%d\n",*p);
  free(p);
  return 0;
}
```

Explain what is wrong with this program.

Q3. Write a C-program that uses a dynamic array of unsigned long long int numbers (8 bytes, only positive numbers) to compute the *n*'th Fibonacci number, where *n* is given as command line argument. For example, **./fib 60** should result in 1548008755920.

Hint: The placeholder %lld (instead of %d) can be used to print an unsigned long long int. Remember that the Fibonacci numbers are defined as Fib(1) = 1, Fib(2) = 1 and Fib(*n*) = Fib(*n*-1)+Fib(*n*-2) for *n*≥3.

Q4. Describe in words how you would implement a *queue* ADT using a dynamic linked list. Which of the functions for the linked list implementation of a stack from the lecture need to be changed, and how?

Q5. Suppose that you have a stack *S* containing *n* elements and a queue *Q* that is initially empty. Describe how you can use *Q* to scan *S* in order to check if it contains a certain element *x*, with the additional constraint that your algorithm must return the elements back to *S* in their original order. You should **not** use an additional array or linked list — only use *S* and *Q*.

Q6. Write a C-program called **llbuild.c** that builds a linked list of integers from user input. The program works as follows:

- starts with an empty linked list called *all* (say), initialised to NULL
- prompts the user with the message "Enter a number: "
- makes a linked list node called *new* from user's response
- appends *new* to *all*
- 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 "Finished. List is " followed by the contents of the linked list in the format shown below.

A sample interaction is shown as follows:

```
prompt$ ./llbuild
Enter an integer: 12
Enter an integer: 34
Enter an integer: 56
Enter an integer: quit
Finished. 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$ ./llbuild
Enter an integer:#
Finished.
```

Q7. Extend the C-program in Q6 to split the linked list in two equally-sized 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: #
Finished. List is 421->456732->321->4->86->89342->9
First half is 421->456732->321->4
Second half is 86->89342->9
```