CSC209 Fall 2016

Week 7 in-class exercise. Linked lists

You are given the following definition of the Island datatype:

```
typedef struct island {
      char * name;
      int population;
} Island;
```

1. Write the program that defines 3 islands:

```
Island one = {"Happy",1000};
Island two = {"Empty",0};
Island three = {"Dense",1000000};
```

How would you represent a tour **one → two → three** using an array?

2. Now we have one more island:

```
Island four = {"Sad", 1, NULL};
```

And we want to change our tour to *one* → *two* → ***four*** → *three*.

How easy it is to dynamically insert a new island in the middle of an array? What data structure would you use instead? What should we add to the definition of Island?

Implement the original tour *one* → *two* → *three* using this new data structure, and insert island *four* after island *two*.

3. Implement function *print_tour* which accepts the head of the linked list as a parameter, and prints all islands in the tour.

4. We want to be able to build our tour dynamically, by reading island information from *stdin*. We will use function *fgets* to read each island name entered from the standard input.
   Simplify island definition. Now each island only stores the name and the pointer to the next:

```
typedef struct island {
      char * name;
      struct island * next;
} Island;
```

Write code for reading island names from *stdin* using *fgets* and print them to *stdout*. The program reads lines until user types "q".

When you run your code what do you notice about *fgets*? Does it include end-of-line characters?

Fix this problem by inserting '\0' instead of end-of-line characters:

```
buffer [strcspn (buffer, "\r\n")] = '\0';
```

5. Read island names from *stdin*, and dynamically add new islands to the tour. After user enters “q”, print islands using the *`print_tour`* function implemented in step 3.

6. Compile your program into executable *islands* with debugging flag `-g`:

```
gcc -g -Wall -std=c99 islands.c -o islands
```

Now test your program for memory leaks with ***valgrind***:

```
valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes ./islands
```

Is the number of *mallocs* equal to number of *frees*?

7. Implement function *`free_islands`* which will free all dynamically allocated list nodes. Call this function before the end of the program.

8. Run *valgrind* again. Does it still complain?
   Replace all calls to *`malloc`* with *`calloc`*, and run valgrind again. This should produce the following reassuring message:

```
==6627== All heap blocks were freed -- no leaks are possible
==6627== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```