Queen Mary
University of London

School of Electronic Engineering and Computer Science

ECS501U – C Programming (2021/22)
Laboratory Session Week_5

Learning Objectives

- To apply adequate functions (e.g. **`strcopy()`**, **`strcat()`**) to programs that process and manipulate strings.
- To dynamically allocate and free memory using appropriate C functions.
- To construct simple linked lists and stacks.

Exercises

You should attempt the exercises below by using only the C constructs that you learnt up to teaching week 6, and:

1. Write pseudo code to describe the required algorithm to solve the exercise (or draw up a flowchart), before writing and testing the actual code.
2. Add comments to your code.
3. Make your code neat, by using indentation and parenthesis (where appropriate).
4. Give meaningful names to functions and variables.

Exercise 1

Write a program that reads in a list of words separated by commas, from the input stream (the keyboard in this case); the program should then extract the words and display each on a separate line, removing any leading or trailing spaces. Save your program to a file called **`wordExtractor.c`**.

Exercise 2

Write a program that reads in a positive integer value less than 100 from the input stream (the keyboard in this case); the program should then generate and display a string corresponding to the integer value in words. For example, if the user enters value **`25`**, the program should display the string **`"Twenty five"`**. Save your program to a file called **`numberToTextConverter.c`**.

Exercise 3

Write code that implements a stack, including the following functions[1]:

- function **`push()`** creates a new node and places it on top of the stack;
- function **`pop()`** removes a node from the top of the stack, frees the memory that was allocated to the removed node and returns the value that was in the removed node;
- function **`isEmpty()`** checks if the stack is empty or not, and thus whether a node can be removed from the stack;
- function **`printStack()`** displays to all the current nodes in the stack to the standard output stream (the screen), after each call to **`push()`** and **`pop()`**.

The functionality above will use a simple node defined as follows:

```
struct stackNode {
  int nodeData;
  struct stackNode *nextPtr;
}
```

Save your code and data structure to a file called **`stack.c`**.

Now add a **`main()`** function to **`stack.c`** that uses the node defined above, to test the code's functionality; your program should allow the user to add and remove a node from the stack, until it decides to terminate the program.

ECS501U – END of LAB 5

[1] You need to consider which arguments and return types these functions should have.