# YSC3217 - MidTerm Assignment - Graph

All your code should be contained in a single ZIP file that will contain both your C and H file (assignment2_yourname.c and assignment2_yourname.h, e.g. `midterm_perrault.c` and `midterm_perrault.h` for me. The program must be compiled using the following flags: `-ansi -pedantic -Wall –Werror`

**The .h file will contain the prototypes of all the functions (except main) as well as the data structures you will use in the program. Do not pull implementations of functions or any other code in the header file.**

In this project, you will build and print graphes. A graph contains a large amount of nodes, connected by edges. For this exercise, our nodes will contain generic data. The creation of the nodes follows an interactive process, in which the program will ask the user to provide information about the data that need to be provided.

You have to build on the following structure to represent your nodes:

```
typedef struct m_node {
      void * data;
      struct m_node ** connected_nodes;
      /* add whatever is relevant here */
} node;
```

Your nodes will store different type of data [either `char *, int` or `float`]. You are not allowed to include anything else to store "values", the value of a node should be stored in the data buffer.

You also need to define a **graph** type. A graph contains a main node (a root). From this root, you should be able to recursively access any node in the graph.

### Prototypes of the functions to implement

`graph * create_new_graph();` Creates a new empty graph.

`node * create_node(graph *g, void * d, …);` **Complete that prototype if needed**. This function creates a new node, without any edges

`void add_node(graph *g);` Adds a node to a graph. This function will prompt the user for any important information, such as: data type [either `char *, int, float`] and value to input from the command line. If the node is the first one in the graph, then it becomes the root.

`void delete_node(graph *g);` Prompts the user to choose a node to delete. This function should display a list of nodes, the user should choose one to delete. Note that deleting a node means that you also need to delete the edges.

`void create_edge(graph *g);` Creates an edge between two nodes. Note: each node should store the edge. Therefore each edge is stored on two nodes.

`void edit_node(graph *g);` Allows the user to choose a node from a list, and change its value.

`void edit_edge(graph *g);` Allows the user to edit a given edge, e.g. changing the edge connecting node 1 to node 2 to an edge connecting node 2 to node 3 instead.

`void delete_edge(graph *g);` Prompts the user to choose a given edge, and deletes it.

`void print_node(graph *g);` Prompts the user to choose a node. The node is printed with every relevant information (value and edges, i.e. connection to over nodes).

`void print_graph(graph *g);` Prints the current graph from the root. Prints each node (with values) and edges (displayed in a text interface). Note: you may print redundant information.

`void distance_nodes(graph *g);` Computes the shortest distance between two nodes. Prompts the user to select two nodes, then computes the shortest distance between them.

`void print_path(graph *g);` Prompts the user to select two nodes and prints the shortest path between them, with each intermediate node (and the value of each node).

`nodes ** get_nodes_at_distance(int distance, graph *g);` Returns all the nodes located at a distance *distance* from the root of the graph. Consider any distance (not only the shortest distances between two nodes): as long as there is a path that goes from the root to this node with a distance of exactly distance. Note that a path cannot cover the same edge twice.

`void delete_graph (graph *g);` Deletes a graph and all the nodes associated with the graph.

`void export_graph(graph *g, char *filename);` Exports the current graph as a XML file containing all relevant information about the current graph (node and edges) to the file *filename*.

`graph * load_graph(char * filename);` Imports a graph from the file *filename* given as an argument. Returns NULL if the file is not an XML file correctly formatted.

### Grading

To grade this project, I will run multiple tests. Each test will be worth some points. I will also double check memory management (make sure to free resources when they are not needed anynore for example).

You are free to do any design choice you want as long as you respect the instructions.