Data Structures & Algorithms Notes

C++ Basics

- Allocation
  - **Static:** Values are automatically deallocated when they go out of scope.
  - **Dynamic:** Values are dynamically added to the heap using the ‘new’ keyword. C++ has no garbage collection so you must deallocate it yourself.
- Datatypes
  - **Null:** In C++, null is defined as the value of 0 or something that looks like it.
  - **Pointer:** Variables that refer to a specific point in memory.

```
// Defining a new pointer
Node* nodePtr;

// Dereferencing a pointer
Node nodeReal = *nodePtr;

// Using the dereferencing operator to use members of object
nodePtr->getValue();
```

  - **Reference:** Similar to pointers, they point to locations in memory. They cannot change where they point to after initialisation but may be used as an object would. They are useful for passing values around without copying them.

```
// Defining a new reference
Node& nodeRef;

// They can be used in exactly the same way as the object
nodeRef += 1;
```

  - **String:** Stored as null-terminated char arrays, kept in the wrapper class of std::string.

```
// Defining a string from a string value
std::string s = “Hello, World!”;
```

- **Array:** In C++, arrays are statically allocated and one must define the length on initialisation.

```
// Defining an array with a set length
int array[3];

// Defining an array from a set of values
int array[] = {1, 2, 3};
```

- **Vector:** The std::vector class is somewhere between a list and an array in functionality as it allows for dynamic length alongside minimal overhead.

```
// Defining a new vector
std::vector vec = new std::vector();
```

- Language Structure
  - **Class:** A template for an object that can have fields and methods C++ .

```
// Defining a class

// Include tags to use the string class
#include <string>

// Using statements allow the program to assume all strings use std library
using std::string;

// Extend parent classes using a colon
Class Node : public ParentClass {

    // Declare scope using the public and private declarations
    private:
        int id;
        string value;

    public:
        string getValue();
        void setValue(string newValue);
        int getId();
        // Very common to abstract method definition from implementation
}
```

- **Header File:** Files used to define classes and methods to create virtual classes they have a .h extension. Source code usually goes into .cpp files.

- **Graph:** A mathematical object that consists of a set of vertices and a set of edges between them. Normally, a vertex will be visualised as a circle, and an edge as a line. Two vertices with an edge between them are **adjacent**, two edges with a vertex between them are **incident**, and a **degree** is the number of edges attached to a vertex.
  Adjacency can be stored as a matrix, along with the vertex nodes, or the whole thing could be object oriented in a graph object.
  To traverse a graph, one must be able to keep track of which vertices have already been visited. This can be done depth-first by recursively visiting an unvisited neighbour, or it can be done breadth-first by iteratively adding unvisited neighbours of each node to a queue of to-visit vertices.

  - **Tree:** A graph with no cycles, meaning that vertexes cannot return to themselves without doubling back on traversal.

    - **Binary Tree:** A tree where each vertex has at most, three neighbours; one parent and two children. Childless vertices are known as leaf nodes.
      They may be traversed either breadth or length-first, where breadth covers each level's nodes before continuing downwards, and length covers each edge, moving across the tree.

      - **Expression Tree:** A binary tree used to represent expressions, such as boolean and algebraic expressions. They can be traversed to provide pre, post and infix notation output for the expression they contain.

      - **Binary Search Tree:** An ordered data structure that allows for fast insertion, removal and lookup. This is done by having values smaller than the current node pushed over to the left child - otherwise being pushed to the right child (depending on implementation).

  - **Directed Graph:** Graph with directionally-bound edges.

  - **Multigraph:** Graph where multiple edges may exist between the same two vertices.

  - **Hyper-Graph:** Graph where edges may attach to multiple vertices.

  - **Other:** Other modifications may be made including the labelling or weighting of vertices or edges. Edges are often stored as a 2D array of booleans, but can easily contain numerical data for weighting values.