Pointers and Dynamic arrays

* Integer size – 4 bytes
* Address-of operator ‘&’
* Dereference operator ‘\*’ – to access or modify the content of the memory location the pointer is pointing to.
* ‘->’ shorthand meaning the member of pointer
* Member access operator for structures or classes

1. Use dot operator
2. Dereference a pointer pointing to the structure
3. Use the ‘->’ shorthand.

* Dangling pointers – point to no valid object
* Null pointers – pointing to nothing. ‘nullptr’
* Array name is a constant pointer variable pointing to the first element of the array
* Use the ‘new’ operator to create a dynamic variable and use a pointer to access it.
* Use ‘delete’ keyword to delete pointers/dynamic variables.
* Memory leak – when you forget to deallocate the memory pointed to by the pointer you just deleted or changed.
* Use ‘delete []’ to free the dynamic array pointed to by your pointer.
* Addition and subtraction on pointers (because they are just addresses).
* Can also compare if two pointers are the same. i.e if they point to the same location.

Linked lists

* A collection of nodes
* Each node stores a piece of data as well as the address of the next node. (except the last node which is a null address).
* Nodes implemented using a struct in c++
* Traverse the linked list using another pointer (maybe called ‘current’), but never with the head pointer or else, your linked list will be lost.
* Process of building a linked list backward:

1. First create a pointer called head which will be a NULL pointer.
2. Create a new node and fill in the required info
3. Let the next pointer (of this current node) point to the beginning of the linked list (Which is the same point the header pointer is pointing to).
4. Update the header pointer to point to the new head of the list.

* Process of building a linked list forward:

1. Create a node whose ‘next’ is a NULL pointer.
2. Also create a head and tail pointer which are NULL pointers
3. Create a new node and fill in the required info
4. New node’s ‘next’ is a pointer pointing to the point the first node’s ‘next’ was pointing to (which is the Null).
5. First node’s ‘next’ as well as tail pointer is updated to point to the info of the new node.
6. Head pointer stays pointing to the info of the first pointer.

* Inserting a node (either at the beginning or after a certain point)
* Deleting a node (either deleting the first node or a node somewhere else in the linked list)
* Searching for a Node.
* Building a sorted linked list by searching for an appropriate place to place a num and inserting it there.
* Deleting an entire list.
* Doubly-linked list and Circularly-linked list
* Printing a linked list in reverse using recursion.