**Keelan Matthews – Practical 1**

**u21549967**

**Task 1**

1. a – stack – it is created **at** run time

b – stack – the pointer itself is created on the stack, the memory it is pointing to is allocated on the heap.

c[10] – stack – it is created at runtime with a fixed size

d – stack – it is created at runtime with a fixed size

e – heap - it is assigned memory **during** runtime (pointer)

f – heap - it is assigned memory **during** runtime (pointer)

g – stack – it is created at runtime

h – stack – it is created at runtime

n – stack – it is created at runtime

1. It would not work as h is a constant variable and cannot be changed later. Assigning it NULL renders it useless.
2. - g is a char type variable, but is being assigned an integer value

* e pointer is being pointed to a value rather than an address in memory
* c array is not a pointer array, but it is being assigned an address

**Task 2**

1. Before the constructor of the derived class.
2. After the destructor of the derived class.
3. After
4. Class A -> Class B -> Class D
5. Class D -> Class B -> Class A

**Task 3**

1. See code
2. This worked because division was performed between two integer types.
3. This worked because addition was performed between two floating point types.
4. This did work because strings can be concatenated without overloading the operator.
5. This did not work because it is not possible to multiply string data types without overloading the multiplication operator.

**Task 4**

1. “15 15” – ptr\_a is dereferenced and assigned the value 15, and ptr\_b points to the same address of ptr\_a. They are both dereferenced in the cout.

“15 4” – ptr\_b is now made to point to an address that stores 4.

“15 15” – ptr\_b now points back to the same address as ptr\_a

“15 15” – ptr\_a is deleted and made to point to the same address as ptr\_b, which is still pointing to the address that ptr\_a was pointing to before deletion. Each “&\*” pair cancels out which leaves ptr\_b being dereferenced.

“*address of ptr\_a* 15” – ptr\_c points to the address of ptr\_a. When dereferenced it returns the address of the pointer. When it is dereferenced twice it returns the value that ptr\_a is pointing to.

**Task 6**