

Swap Function Exercises

- Briefly describe `swap()`
 - `swap()` efficiently exchanges the values of its arguments
- What exception guarantee does it offer?
 - `swap()` offers the no-throw guarantee
- Why is it often better to use an overloaded version (if available) than the generic version?
 - The overloaded version has more information about the type being swapped and will therefore be better optimized

- Write a simple program that uses `swap()` to exchange the values of two `int` variables
- Print out the values of the variables before and after the `swap()` call
- Write another program in which the code to exchange the values is written out explicitly

- Briefly describe the overloaded swap() for std::string. Why is this more efficient than the generic version?
 - The overloaded swap() can swap the pointers to the memory buffers
 - The generic version would call the copy constructor
 - This needs to allocate a new buffer and copy the data, which would be much slower

- In what situation is it worth overloading swap() for our own class?
 - We expect instances of our class to exchange values, and the copy constructor would be inefficient
 - If we want to provide the strong exception guarantee when assigning instances

- What factors should we consider when writing the overloaded version?
 - It should be defined outside the class, as a global function
 - It should be inline
 - It should either be a friend of the class or call a member function which gives it access to the data members
 - Implement it by calling the overloaded version of `std::swap()` for each data member

- Implement an overloaded swap function for the following class

```
class BufferManager {  
    private:  
        int size;  
        char *buffer;  
};
```