

CS2401**Lab Assignment 1**

This lab explores two important aspects of dynamic memory management. The first part deals with “**The Big Three**,” the destructor, copy constructor and assignment operator that need to be overloaded for any class that holds dynamic memory. The second part deals with the *bad_alloc* exception and how it can be handled.

Part 1:

Begin by making a separate directory for this assignment. In that directory you will be using the main that I have given you and creating a file that will have a class, both the class declaration and the function implementations. (For this lab function implementations can be done under the class declaration, in the same file.)

The class, *Numbers*, that you declare will have these private variables:

```
unsigned long *data;  
size_t used;  
size_t capacity;
```

The functions should be:

1. A default constructor that sets up an initial array of 5 unsigned longs, sets capacity to 5 and used to 0;
2. An `add` function that receives an unsigned long as a parameter and puts it into the next available spot.
3. A `resize` function that is only called when the `add` function checks for and discovers that `used == capacity`. It should increase the size of the array by five.
4. A `remove_last` function that takes out the last thing added to the array. (All you need to do for this is to decrement the *used* counter. You do **not** need to change the capacity.)
5. A `display` function that prints out all the numbers in the array, separated by spaces.

These functions have been declared in the class declaration and implemented immediately below that.

In the main program, "`numbers.h`" has been included at the top.

Main Program:

- Creates two objects (`n1`, `n2`)
- Adds some values to `n1`
- displays `n1`
- Makes a copy of `n1` and stores it in `n2`
- Removes few values from `n2`
- Adds new values to `n2`
- Displays `n1` again

On your **Answer sheet** write the answers to the following:

1. What do you see?
2. Is this a problem and why?
3. What caused this to happen?

Implement the overloaded assignment operator for the Numbers class. Remember that an assignment operator has three parts:

- check for self-assignment,
- delete the existing array
- create a new array the same size as the one you're copying from, copy the values for used, capacity and then copy all the data from the other array into this new one. (The new one will be held by the main pointer of the object of which this operator is a member.)

Compile and run the program again without changing anything in the main.

On your **Answer sheet** write the answers to the following:

4. What do you see?
5. Is it different from what you saw before?
6. What caused this to happen – why is it different?

Part 2:

Now, in the main, uncomment the section for part 2.

Note that `reveal_address` is NOT something that we would normally do - the application programmer has no reason to need to know the address of the dynamic array, but I have included it here to help you see what is going on.

On your **Answer sheet**:

7. Write down the five addresses that are output and the byte count output.
8. Can you tell how many bytes they are apart? Write down your best estimate. (Remember that these addresses are appearing in hexadecimal.).

Eventually this array would eat up all of the computer's memory and a `bad_alloc` exception would be "thrown" (and caught). Because modern computers have so much memory, we might wait a very long time for this to happen.

You can experiment by adding a larger number of items to `n3`. Change 100 to 100000 or 300000 and see what happens.

The solution is to write a destructor for the Numbers class. Add a destructor now and in it you should include, in addition to the correct `delete` command and some message you want to print:

```
bytecount = bytecount - capacity*sizeof(unsigned long);
```

Compile and run the program again.

On your **Answer sheet**:

9. What addresses did you see this time and how much are they apart from each other?
10. What is the `byte_count`?
11. Explain why adding the destructor resulted in this different behavior.

Please submit your source code and your answer sheet.

Grading (30 Points):

- 2 points for each of the questions
- 4 points for the overloaded assignment operator
- 4 points for the destructor