MT 2MD3 Assignment 1 – Michael Giancola

1. [short answer, 3 marks]: Declare A to be a pointer to integer and assign a value of 21 to its referent. How would you write an expression whose value is twice the value of A's referent?

```
int val = 21; //sets value to 21
int *A = &val; //referent of A is set equal to address of value
int newVal = 2*(*A); //new variable is equal to 2 times the referent value
of A
```

2. [short answer, 2 marks]: Consider the following attempt to allocate a 10-element array of pointers to doubles and initialize the associated double values to 0.0. Rewrite the following (incorrect) code to do this correctly. (Hint: Storage for the doubles needs to be allocated.)

```
double* dp[10]; //creates an array of pointers to doubles
for (int i = 0; i < 10; i++) //for loop to run through array of pointers
{
    dp[i] = new double; //allocate new memory at each pointer
    *(dp[i]) = 0.0; //set value of each pointer to 0.0
}
```

The provided example does not allocate any memory for each pointer in the array. This is why you must use the new component to allocate memory.

3. [short answer, 1 mark]: What (if anything) is different about the behaviour of the following two functions f and g that increment a variable and print its value?

```
void f(int x)
{ std::cout << ++x; }

void g(int& x)
{ std::cout << ++x; }
```

In the first function "f(int x)", the variable x is passed by value, meaning that a copy of the value of the variable that passes through the function is made and used within the function. The increment operator "++x" doesn't increment the original but increments the copy. Therefore, the incremented value of this copy is the output. Whereas in the second function "g(int &x)" is passed by reference rather than value meaning the function doesn't use the copy and uses the actual variable. The increment operator increments the original variable in this case, with the incremented value of the original variable being the output. Therefore, the function g changes the original variable's value after the function call whereas, in the function f, the original variable has no change in value.

4. [short answer, 4 marks]: Write a short C++ function that takes a positive double value x and returns the number of times we can divide x by 2 before we get a number less than 2.

```
int fun(double x)
{
    int count = 0; //set a counter variable
    while (x*0.5 >= 2) //use while loop to check the condition
    {
        x /= 2; //modify the x value
        count++; //increment count
    }
    return count; //return value of count
}
```

5. Write a short C++ function to compute GCD(n, m) for two integers n and m.

```
int gcd(int n, int m)
{
    while (m != 0) //when m is equal to zero it is GCD
    {
        int temp = m; //set variable to m
        m = n % m; //sets new value to m as remainder of dividing these two
values
        n = temp; //set n to the temporary variable
    }
    return n; //returns our GCD
```

7. Suppose we have a variable p that is declared to be a pointer to an object of type Progression using the classes of Section 2.2.3. Suppose further that p actually points to an instance of the class GeomProgression that was created with the default constructor. If we cast p to a pointer of type Progression and call p->firstValue(), what will be returned? Why?

If p is a pointer to an instance of the class GeomProgression that was created with the default constructor and we cast p to a pointer of type progression while calling p->nextValue(), the nextValue() function that is part of the GeomProgression class will be called. The virtual keyword allows for the derived class function to be called instead of the base class. The GeomProgression class inherits from the Progression class and takes over the nextValue() function. This means that when the pointer of type Progression (p) is used to call the nextValue() function, the overridden version in the GeomProgression class will be called.